Université de Montréal

Vérification temporelle des systèmes cycliques et acycliques basée sur l'analyse des contraintes

par

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maitrise
en informatique

Août, 2011

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé :

Vérification temporelle des systèmes cycliques et acycliques basée sur l'analyse des contraintes

Présenté par :
Ahmed Azzabi

–

a été évalué par un jury composé des personnes suivantes :

Michel Boyer, président-rapporteur
El Mostapha Aboulhamid, directeur de recherche
Gabriela Nicolescu, co-directeur
Bernard Gendron, membre du jury

Résumé

Nous présentons une nouvelle approche pour formuler et calculer le temps de séparation des événements utilisé dans l'analyse et la vérification de différents systèmes cycliques et acycliques sous des contraintes linéaires-min-max avec des composants ayant des délais finis et infinis. Notre approche consiste à formuler le problème sous la forme d'un programme entier mixte, puis à utiliser le solveur Cplex pour avoir les temps de séparation entre les événements. Afin de démontrer l'utilité en pratique de notre approche, nous l'avons utilisée pour la vérification et l'analyse d'une puce asynchrone d'Intel de calcul d'équations différentielles. Comparée aux travaux précédents, notre approche est basée sur une formulation exacte et elle permet non seulement de calculer le maximum de séparation, mais aussi de trouver un ordonnancement cyclique et de calculer les temps de séparation correspondant aux différentes périodes possibles de cet ordonnancement.

Mots-clés : Vérification et analyse des systèmes, temps de séparation des événements, contraintes linéaires-min-max, système cyclique et acyclique.

Abstract

We present a new approach for formulating and computing time separation of events used for timing analysis of different types of cyclic and acyclic systems that obey to linear-min-max type constraints with finite and infinite bounded component delays. Our approach consists of formulating the problem as a mixed integer program then using the solver Cplex to get time separations between events. In order to demonstrate the practical use of our approach we apply it for the verification and analysis of an Intel asynchronous differential equation solver chip. Compared to previous work, our approach is based on exact formulation and it allows not only the maximum separation computing, but it can also provide cyclic schedules and compute bound on possible periods of such schedules.

Keywords: System verification and analysis, Time separation of events, Linear-min-max constraints, Cyclic and acyclic systems

Table des matières

Liste des figures

Liste des tableaux

Acronymes:

mmlp: min-max-linear problem

MILP: programme linéaire en nombres entiers mixtes

CAS : Cyclic Approximatif Séparations

À ma mère

Remerciements

Tout d'abord, je tiens à remercier et exprimer ma gratitude à mon directeur de recherche, le professeur El Mostapha Aboulhamid pour son énorme soutien et ses précieux conseils ainsi qu'à ma co-directrice le professeur Gabriela Nicolescu. Je remercie aussi le professeur Bernard Gendron (CIRRELT) pour ses conseils et son aide avec le solveur CPLEX ainsi que les membres du laboratoire LASSO, spécialement Alena Tsikhanovich et Amine Anane pour leur support et conseils.

Un grand merci à tous ceux qui ont participé de près ou de loin à la réalisation de ce travail, spécialement ma famille qui m'a soutenu tout au long de ce parcours et m'a encouragé à poursuivre mes études et à qui je dédie ce mémoire.

Enfin, j'adresse mes salutations les plus sincères aux membres du jury qui m'ont honoré en participant à l'évaluation de ce mémoire.

Chapitre I Introduction

De nos jours, les systèmes numériques sont de plus en plus complexes ce qui demande plus d'efforts pour la validation et la vérification. La validation d'un système concerne l'aspect temporel et l'aspect fonctionnel. Dans ce mémoire, nous nous intéressons uniquement à la validation des contraintes temporelles d'un système.

Le facteur temps est crucial pour évaluer les performances d'un circuit et garantir sa compétitivité sur le marché, d'où l'importance de la vérification des contraintes temporelles pour la conception de systèmes numériques fiables répondant aux exigences requises.

1 Définition du problème

Le problème de vérification temporelle peut être formulé de la manière suivante :

Étant donné un système constitué de plusieurs composants interconnectés, ayant chacun des délais avec une certaine incertitude due à l'environnement ou à la fabrication (souvent représentés par une borne inférieure et une borne supérieure), nous cherchons à déterminer si tout le système satisfait les contraintes et exigences requises. Dans le cas de violation de contraintes, nous cherchons aussi à localiser la source de l'erreur et à quel niveau (composant) elle s'est produite.

2 Motivation

La simulation est souvent utilisée pour la vérification et la validation des contraintes temporelles, mais elle n'est pas pratique pour des systèmes de grande taille (le simulateur SPICE peut passer des semaines et même des mois de simulation [1]). D'où la nécessité d'une approche analytique permettant d'accélérer la vérification et par conséquent tout le processus de conception et de mise sur le marché. L'analyse des contraintes temporelles peut être utilisée pour résoudre plusieurs problèmes tels que:

- « Vérification temporelle » : le problème consiste à déterminer si un système avec des contraintes temporelles données (fréquence d'horloge et délais des composants prédéfinis) est consistant (fonctionnera correctement).
- « Optimal Clocking »: le problème consiste à trouver la fréquence d'horloge et l'ordonnancement optimal étant donné un ensemble de contraintes temporelles (cela mène à chercher le cycle d'horloge et la position de ses fronts montant et descendant)[2].
- « Timing driven design » : le problème consiste à essayer de trouver une meilleure façon de concevoir un circuit tout en maintenant le système consistant (les contraintes satisfaites). Souvent les concepteurs commencent par concevoir un système consistant puis ils essaient de l'améliorer pour utiliser moins de ressources ou consommer moins d'énergie tout en maintenant la consistance du système [2].

Les problèmes illustrent l'importance de l'approche analytique. En effet, plusieurs approches et algorithmes ont vu le jour surtout pour les systèmes acycliques avec différents types de contraintes (min, max, linéaires)[1]. Gahlinger [3] a énoncé que pour modéliser les contraintes des systèmes modernes, tous les trois types de contraintes sont requis. Malheureusement, l'optimisation avec les contraintes de types min-max-linéaires a été prouvé comme étant NP-complet[2] [4]. De ce fait, plusieurs chercheurs se sont limités à trouver des algorithmes polynomiaux ou pseudo-polynomiaux pour traiter des systèmes avec uniquement un ou deux types de contraintes, ce qui limite les types de systèmes couverts par ces algorithmes. Pour le cas général avec les trois types de contraintes, uniquement des heuristiques ayant une complexité exponentielle en pire cas ont été proposées. Pour les systèmes cycliques, on trouve beaucoup moins de travaux, vu la complexité qu'ajoute le comportement itératif et les cycles aux problèmes déjà NP-complet avec les trois types de contraintes. Le travail le plus intéressant qu'on a trouvé et sur lequel d'autres travaux récents se sont basés est celui de Chakraborty *et al.*[5] dans lequel ils ont proposé un algorithme approximatif pseudo polynomial limité à un certain type de systèmes et traitant uniquement les systèmes avec deux types de contraintes (`min` et `max`) et de hauteur limitée à un (détails dans les prochains chapitres).

[1] Les types de contraintes seront détaillés dans le prochain chapitre

[2]Notre problème est un problème d'optimisation et non pas un problème de décision mais pour rester cohérent avec les travaux précédents et par convention, dans la suite de ce mémoire on désigne par NP-complet le problème d'optimisation de maximum de séparation.

3 Contribution

Dans ce mémoire, nous proposons une nouvelle formulation du problème permettant le calcul exact des temps de séparation entre les événements des systèmes cycliques et acycliques avec des contraintes min-max-linéaires. Cela nous a permis aussi de calculer la période (durée d'une itération d'opérations) des systèmes cycliques et d'obtenir un ordonnancement correspondant. Notre contribution réside dans le fait que notre approche n'est pas limitée aux graphes acycliques, mais elle couvre aussi les systèmes cycliques avec tout type de contraintes sans faire le dépliage de graphe représentant les contraintes temporelles. En plus, mise à part la résolution du problème de vérification temporelle, notre approche a permis de résoudre d'autres problèmes connexes liés à l'analyse temporelle comme celui de l'ordonnancement des opérations des systèmes cycliques [6].

Notre approche est basée sur la transformation du problème de vérification temporelle en un problème d'optimisation sous forme de programme linéaire en nombres entiers mixtes (MILP). Nous avons utilisé le solveur Cplex afin d'obtenir les temps de séparations. De plus, pour vérifier et montrer l'efficacité de notre approche en pratique, nous l'avons appliquée sur l'exemple réel d'une puce asynchrone de résolution d'équations différentielles conçue par Intel [5], afin de vérifier les contraintes temporelles et obtenir les temps de séparation correspondants. Nous avons réussi à avoir les mêmes temps de séparation que [5] en quelques secondes et à déceler une erreur lors de la conception pouvant causer une violation de contraintes et le mal fonctionnement de la puce. Les résultats de notre recherche ont été présentés au cours de la conférence ICECS 2010 [7].

Le chapitre suivant sera consacré à l'étude de la vérification temporelle pour les systèmes acycliques. Nous présentons la modélisation sous forme de graphes, ainsi que la terminologie et les concepts relatifs aux problèmes de vérification et d'analyse temporelle (les types de contraintes, notion de consistance, notion de hauteur, maximum de séparation, etc.). Dans le chapitre 3, nous nous intéressons à la vérification temporelle pour les systèmes cycliques, qui constitue le sujet principal de ce mémoire. Nous présentons le graphe utilisé dans ce mémoire, ainsi que la formulation générale du problème sous forme de problème d'optimisation.

Dans le quatrième chapitre, nous présentons notre solution pour résoudre le problème de maximum de séparation pour les systèmes cycliques. Nous décrivons les différentes étapes de transformation du problème de maximum de séparation en un programme linéaire en nombres

entiers mixtes qui sera résolu par la suite avec le solveur Cplex; à la fin de ce chapitre nous décrivons l'outil développé. Le chapitre 5 couvre les applications et expérimentations effectuées sur différents graphes et systèmes cycliques et acycliques, ainsi que les résultats obtenus en les comparants avec les travaux précédents. Nous présentons toutes les étapes de l'approche analytique à travers l'exemple de la puce d'équation différentielle, en commençant par la modélisation graphique des composants sous forme de graphes à partir de la description RTL, puis la formulation des exigences de l'environnement, et finalement, le calcul du maximum des temps de séparation et validation. Le chapitre 7 conclut ce mémoire, en donnant une idée sur les limites et les travaux futurs possibles.

Chapitre II Vérification temporelle des systèmes acycliques

Au cours de ce chapitre, nous nous intéressons au problème de vérification temporelle pour les systèmes acycliques. Tout d'abord, nous introduisons le problème de vérification temporelle et sa représentation sous forme de graphe. Nous mettons l'accent sur les différentes notions et termes (tels que les contraintes et leurs types, etc.) qui lui sont liées et qui seront nécessaires à la compréhension de la suite de ce mémoire. Ensuite, nous donnons la formulation du problème de séparation maximum sous forme d'un programme en nombres entiers. Puis, nous terminons ce chapitre avec une étude de complexité du problème de maximum de séparation pour les systèmes acycliques en passant en revue quelques solutions proposées dans des travaux précédents.

1 Modélisation du problème sous forme de graphe

Tout d'abord, rappelons les objectifs du problème de vérification temporelle des systèmes numériques cité précédemment dans l'introduction. Dans cette section, nous nous limitons aux systèmes acycliques qui vont être définis un peu plus loin.

Objectif du problème de vérification temporelle:

Nous essayons de répondre à deux questions :

1. Étant donné un système qui doit obéir à un ensemble de contraintes temporelles, est-ce que cet ensemble de contraintes ne contient pas de contradictions? Dans ce cas, nous dirons que le système est consistant.
2. Au cas où on doit intégrer ce système dans un environnement, est-ce que le système satisfait les exigences temporelles imposées par cet environnement?

Avant de pouvoir effectuer l'analyse et la vérification temporelle proprement dites, le comportement du système numérique est décrit par un graphe d'événements construit à partir d'autres descriptions standard tel que les chronogrammes et les schémas RTL. En effet, les chronogrammes sont souvent utilisés vu qu'ils sont simples et reflètent bien le comportement externe d'un circuit. Un chronogramme montre les transitions et les changements des valeurs des signaux d'un circuit; ces changements sont appelés événements. En plus des événements,. les chronogrammes permettent de représenter les caractéristiques et contraintes temporelles qui définissent les relations entre les événements. Chacune de ces contraintes peut être représentée par un délai avec une borne inférieure et une borne supérieure. À chaque chronogramme, on peut associer un graphe d'événements dont les nœuds représentent les événements et les arcs représentent les délais (contraintes). Un exemple de chronogramme est donné dans la Figure 1 et le graphe d'événements, correspondant est illustré dans la Figure 2. Notons que, dans cette illustration, nous supposons que toutes les contraintes sont linéaires (ceci est expliqué plus tard).

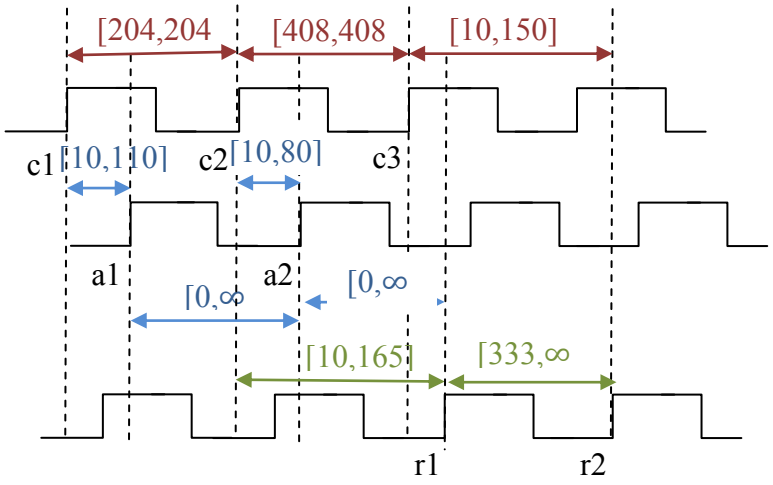


Figure 1 Chronogramme partiel de cycle de lecture du processeur 8086

.

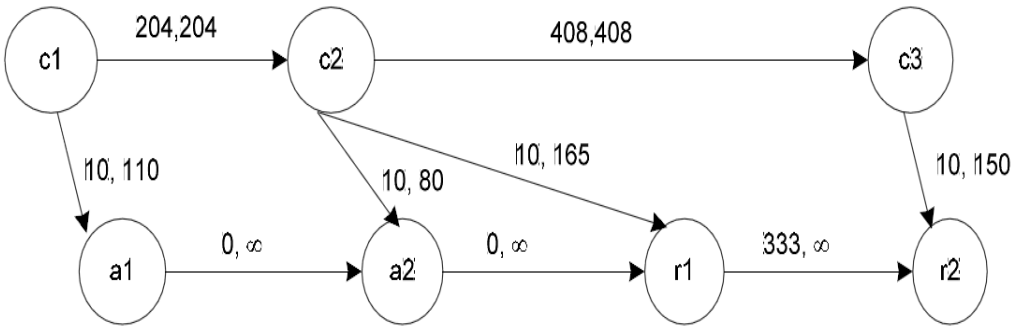
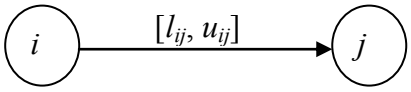


Figure 2 Graphe d'événements partiel correspondant à la Figure 1

Définition 2.1 : Un graphe d'événements $EG = (E, V)$ est un graphe acyclique où E est l'ensemble des événements et V l'ensemble des dépendances représentées sous forme d'arcs, chaque arc v_{ij} reliant l'événement i à l'événement j. À chaque arc v_{ij} on associe une étiquette sous forme d'un intervalle $[l_{ij},u_{ij}]$ où les bornes sont des entiers. À chaque événement on associe une étiquette $T(i) \in \{\text{lin, min, max}\}$ (définie plus tard).

Définition 2.2 : Un graphe d'événements, ou par extension, le système qu'il décrit, est dit consistant si on peut associer à tout événement i un entier naturel $t(i)$ désignant le temps d'occurrence de l'événement i, tel que toutes les contraintes décrites par le graphe sont satisfaites

Graphiquement, dans le cas où un événement a un seul prédécesseur :



signifie que: $t(i) + l_{ij} \leq t(j) \leq t(i) + u_{ij}$

Dans le cas où l'événement est précédé par plusieurs événements, on lui associe un type de contrainte qui déterminera l'intervalle de son temps d'occurrence et sa relation par rapport à ces prédécesseurs. Vanbekbergen *et al.* [8] ont défini une interprétation graphique des trois types de contraintes et Gahlinger [3] a stipulé que les trois types sont nécessaires pour modéliser les systèmes numériques et résoudre le problème de vérification temporelle.

Les différents types de contraintes sont :

- Linéaire: toutes les contraintes (bornes inférieures et supérieures) entre l'événement et tous ces prédécesseurs doivent être satisfaites. Cet événement est étiqueté dans le graphe par T(i) = lin.
- Max : Par opposition au type linéaire, toutes les bornes inférieures et uniquement la borne supérieure maximale (la plus grande) des contraintes reliant un événement à ces prédécesseurs doivent être satisfaites. Cet événement est étiqueté dans le graphe par T(i)= max.
- Min : Toutes les bornes supérieures et uniquement la borne inférieure minimale (la plus petite) des contraintes reliant un événement à ces prédécesseurs doivent être satisfaites. Cet événement est étiqueté dans le graphe par T(i) = min.

On peut écrire les différents types de contraintes sous la forme des inéquations suivantes:

- Linéaire :

$$\max_{i \in pred(j)} (t(i) + l_{ij}) \leq t(j) \leq \min_{i \in pred(j)} (t(i) + u_{ij})$$

- Max :

$$\max_{i \in pred(j)} (t(i) + l_{ij}) \leq t(j) \leq \max_{i \in pred(j)} (t(i) + u_{ij})$$

- Min :

$$\min_{i \in pred(j)} (t(i) + l_{ij}) \leq t(j) \leq \min_{i \in pred(j)} (t(i) + u_{ij})$$

*Avec $pred(j) = \{i | v_{ij} \in E\}$, E est l'ensemble des arcs du graphe

Un exemple et une interprétation graphique des trois types de contraintes sont donnés par la Figure 3, dans laquelle l'événement *D* est précédé par les événements *A, C* et *B*.

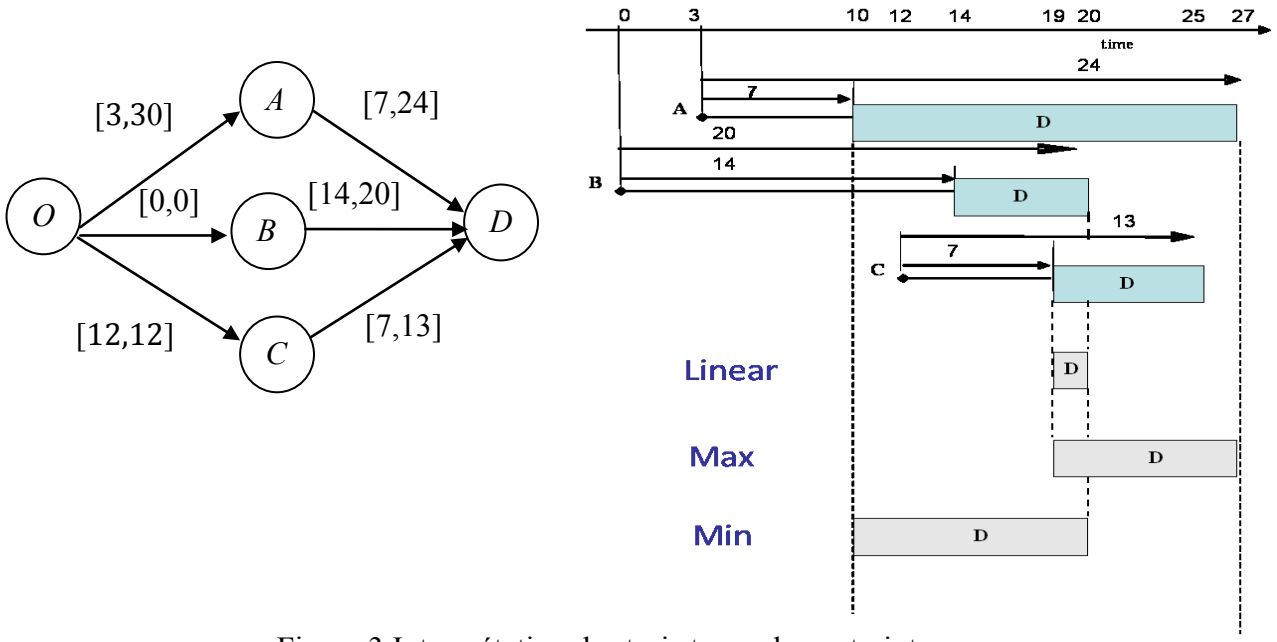


Figure 3 Interprétation des trois types de contraintes

On a *D* précédé par *A, B* et *C* donc on aura :

$$t(A) +7 \leq t(D) \leq t(A) +24$$

$$t(C) +7 \leq t(D) \leq t(C) +13$$

$$t(B) +14 \leq t(D) \leq t(B) +20$$

Si *D* est de type linéaire, alors il doit satisfaire toutes les contraintes des bornes inferieures et supérieures; on obtient :

$$\max (t(A) +7, t(C) +7, t(B) +14) \leq t(D) \leq \min(t(A) +24, t(C) +13, t(B) +20)$$

$$\Rightarrow \quad 19 \leq t(D) \leq 20$$

De la même manière si on considère *D* de type max alors on obtient :

$$\max (t(A) +7, t(C) +7, t(B) +14) \leq t(D) \leq \max(t(A) +24, t(C) +13, t(B) +20)$$

$$\Rightarrow \quad 19 \leq t(D) \leq 27$$

De même si on considère *D* de type min on obtient :

$$\min (t(A) +7, t(C) +7, t(B) +14) \leq t(D) \leq \min(t(A) +24, t(C) +13, t(B) +20)$$

$$10 \leq t(D) \leq 20$$

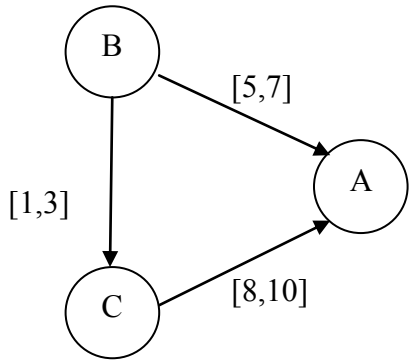
2 Consistance et maximum de séparation

Dans l'énoncé du problème de vérification et analyse temporelle, on a souligné que l'objectif de la vérification est de déterminer si tout le système est consistant et s'il satisfait les exigences de l'environnement. Tout d'abord, nous revisitons la notion de consistance déjà définie dans la section précédente.

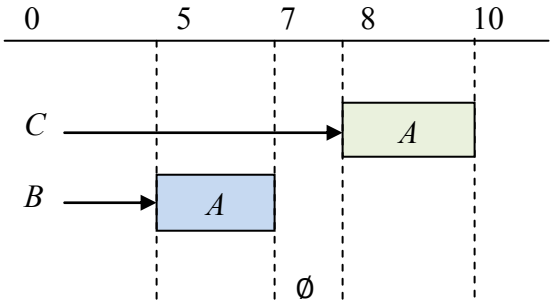
2.1 Vérification de la consistance

Nous disons qu'un graphe d'événements est consistant s'il admet au moins une solution pour le système d'inéquations correspondant.

Prenons cet exemple où toutes les contraintes sont linéaires :



$$S \begin{cases} t(C) + 8 \leq t(A) \leq t(C) + 10 & (1) \\ t(B) + 5 \leq t(A) \leq t(B) + 7 & (2) \\ t(B) + 1 \leq t(C) \leq t(B) + 3 & (3) \end{cases}$$



⇨ $S' \begin{cases} 8 \leq t(A) - t(C) \leq 10 & (1) \\ 4 \leq t(A) - t(C) \leq 4 & (2) - (3) \end{cases}$

⇨ *S* n'admet pas de solution

En pratique, dans une étude d'un système numérique, si l'événement *A* modélise un signal qui se déclenche à la réception des signaux *B* et *C*, la vérification de la consistance permettra de détecter que le signal *A* ne pourra jamais avoir lieu, ainsi que tous les signaux qui en dépendent.

Ceci permettra de détecter une anomalie qui pourra causer un mauvais fonctionnement du système conçu, d'où l'importance de la vérification de la consistance.

2.2 Maximum de séparation

Jusqu'à présent, nous n'avons considéré que le cas où le système à concevoir est soumis uniquement à des contraintes temporelles imposées par les ressources et les composants du système ou une spécification initiale qu'on appelle contraintes de conception; par exemple, nos ressources ne nous permettent pas de dépasser une cadence d'horloge de 2 Mhz. Mais une fois que le système conçu est intégré dans un environnement, cela nous impose de nouvelles exigences qui sont formulées sous forme de contraintes temporelles (intervalle de temps) que nous appelons contraintes d'environnement; par exemple, un débit de résultat chaque microseconde. Ceci nous ramène au deuxième objectif de la vérification temporelle après la vérification de la consistance, qui consiste à s'assurer que le système conçu satisfait les exigences d'intégration (satisfait les contraintes d'environnement). Afin d'atteindre cet objectif, l'une des méthodes les plus utilisées est le calcul du maximum de séparation qui consiste à calculer le temps maximum qui sépare le temps d'occurrence d'une paire d'événements. Par exemple, une contrainte de l'environnement peut imposer que le temps qui sépare l'occurrence du signal k et du signal l, soit dans l'intervalle [10,50]. Pour vérifier que c'est le cas, nous pouvons calculer le maximum des temps de séparation entre cette paire d'événement et vérifier si les contraintes d'environnement sont satisfaites en les comparant avec les séparations maximales obtenues.

Le problème de maximum de séparation peut être mis sous la forme d'un problème d'optimisation dans lequel on cherche à maximiser la différence entre les temps d'occurrence de chaque paire d'événements sous les différents types de contraintes.

Formulation du problème de séparation maximum pour les systèmes acycliques

$$Max.\, t(k) - t(l)\,, \qquad k\ et\ l\ donnés$$

$$\begin{cases} t(j) - t(i) \le u_{ij}\,, i \in pred(j), T(j) = lin\ ou\ T(j) = min \\ t(i) - t(j) \le -l_{ij}\,, i \in pred(j), T(j) = lin\ ou\ T(j) = max \\ t(j) \le \max(t(i) + u_{ij})\,, i \in pred(j), \qquad T(j) = max \\ \min(t(i) + l_{ij}) \le t(j)\,, i \in pred(j), \qquad T(j) = min \end{cases}$$

3 Complexité et revue de littérature

La complexité de résolution du problème de séparation maximum dépend essentiellement des types de contraintes.

McMilan et Dill [4] ont prouvé à l'aide d'une réduction du problème 3-Sat (Figure 4), que le cas général de recherche de séparation maximale avec des contraintes min-max-linéaires est NP-complet (nous donnerons une preuve plus simple plus loin dans ce chapitre).

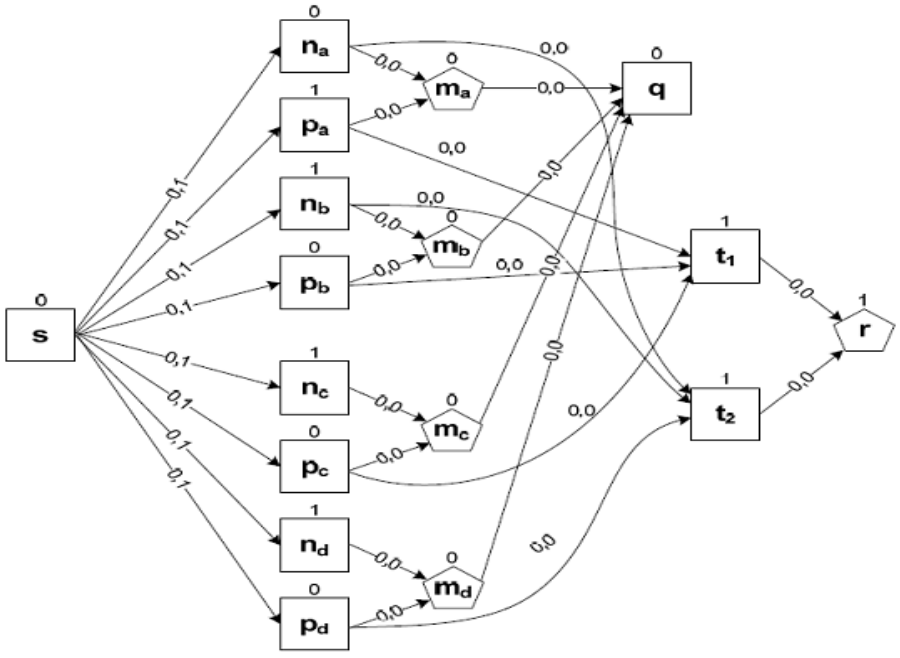


Figure 4 Réduction 3-Sat du problème Min/Max (les carrés sont des Max et les pentagones sont des Min) [4]

Ils ont aussi proposé un algorithme polynomial pour résoudre le problème avec des contraintes max uniquement et un autre algorithme pseudo-polynomial pour le cas avec des contraintes `max` et `linéaires`. Pour le cas général, celui avec des contraintes min-max-linéaires, ils ont proposé une heuristique basée sur une adaptation de l'algorithme *branch and bound* en éliminant récursivement les contraintes `min` afin de générer des sous-problèmes de type max-linéaires uniquement. Les résultats ont montré que leur approche est assez efficace en

pratique, mais reste toujours exponentielle au pire des cas; de plus, leur algorithme peut prendre un temps exponentiel pour détecter un faux cycle négatif [9].

T. M. Burks et K. Sakallah [2] ont aussi proposé une manière plus simple de montrer que le problème avec les contraintes min-max-linéaire est NP-complet, pour les systèmes acycliques. Ils utilisent une réduction du problème 0-1 IP (connu comme étant NP-complet) à une instance du problème min-max-linéaire :

Théorème :mmlP (min-max-linear problem, équivalent à notre formulation du problème de séparation maximum donnée précédemment) est NP-complet[2]

Preuve : d'abord on définit un mmLP ayant les mêmes contraintes linéaires que 0-1IP.

Pour chaque variable entière x_i ajouter max (x_i, 1-x_i)=1. L'unique solution de ce problème est x_i= 1oux_i = 0. Ce qui complète la réduction. Pour les contraintes `min` une équation équivalente est `min` (x_i, 1-x_i) =0, notons aussi qu'il suffit de multiplier par -1 les deux côtés de l'équation, et ajouter 1 pour obtenir :

$$\min(x1, x2, \quad \ldots, xn) = -\max(-x1, -x2 \ldots, -xn)$$

Donc, dans tous les cas il est possible de représenter chaque variable 0-1 dans une forme de mmLP. D'où le problème général mmLP est NP-complet (vu que 0-1 IP est NP-complet).

Après avoir prouvé que le problème de mmLP est NP-complet, ils ont proposé deux méthodes pour le résoudre :

- La première repose sur une optimisation de la méthode *branch and bound* pour réduire le nombre d'opérations requis par l'algorithme de programmation linéaire Simplex.
- La deuxième approche consiste à transformer le problème sous forme de MILP (mixed integer linear program) et après utiliser un solveur pour le résoudre.

Les résultats des tests sur des systèmes ISCAS [10] Benchmarks ont montré qu'effectivement leur méthode avec *branch and bound* requiert moins d'opérations. Néanmoins, la résolution avec le solveur *lp_solve* [2] donne de meilleurs résultats (en terme de temps de calcul) ceci étant probablement dû à l'implémentation optimisée du solveur.

Yen, Ti-Yen et *al.* [9] ont présenté un algorithme pour le calcul de temps de séparation entre les événements dans un graphe acyclique. En premier lieu, ils ont élaboré un algorithme

pour le cas max-linéaire. La complexité de cet algorithme n'a pas été vérifiée, mais elle est conjecturée à $O(V^2 \log V + VE)$ [9]. Après ils ont utilisé une heuristique basée sur une adaptation de l'algorithme de *branch and bound* afin que leur algorithme pour les contraintes max-linéaire puisse résoudre le cas avec des contraintes min-max-linéaire.

L'avantage majeur de cette solution réside dans le fait qu'elle a permis de contourner le problème de faux cycles négatifs dont souffre l'algorithme de McMilan & Dill [4]. Par la suite, afin de résoudre le problème plus général avec des contraintes min-max-linéaire, ils ont modifié leur algorithme pour les contraintes max-linéaires en utilisant une adaptation de l'algorithme de *branch and bound* un peu similaire à celle utilisée par McMilan & Dill [4] (division du problème principal en sous problème en effectuant des relaxations successives des contraintes de type `min`). L'algorithme est bien détaillé dans [9].

Dernièrement en 2009, A. Tsikhanovich et E. M. Aboulhamid ont proposé une solution basée sur la linéarisation des contraintes min/max [11]. Par la suite, ils ont utilisé une adaptation de *branch and bound* inspirée de l'algorithme de Yen, Ti-Yen *et al.*[9] pour réduire le nombre d'explorations en éliminant les bornes supérieures des contraintes `max` et les bornes inférieures pour les contraintes `min`, étant donné que pour les contraintes `max` et `min`, uniquement une borne qui doit être satisfaite. Ensuite, ils ont essayé d'intégrer leur algorithme dans différents niveaux d'abstraction TLM. Les résultats ont montré que la méthode de *branch and bound* a permis de réduire le nombre d'explorations. Toutefois, les tests que nous avons effectués, dès que le graphe grandit, on commence à obtenir des valeurs erronées et le temps de résolution devient exponentiel, car dès l'ajout de quelques nœuds min/max, le temps nécessaire pour calculer les temps de séparation augmente considérablement.

Le tableau suivant résume les différents algorithmes et leurs complexités selon les différents types de contraintes pour les systèmes acycliques. Notons que, jusqu'à présent, nous n'avons traité que les systèmes acycliques; le reste du mémoire se consacrera aux systèmes cycliques.

TABLEAU 1 COMPLEXITÉ DU CALCUL DU MAXIMUM DE SÉPARATION SELON DIFFÉRENTS ALGORITHMES AVEC DIFFÉRENTS TYPES DE CONTRAINTES POUR LES SYSTÈMES ACYCLIQUES

Type de contrainte	**Complexité**
Linéaire uniquement	*O(VE) (algorithme de plus court chemin)*
Max uniquement	*O(E)* [1]
Max+Linéaire	$O(V^2 \log V + VE)$, conjecture [3]
Min+Max	NP-complet [1]
Min+Max+Linéaire	NP-complet [4]

4 Conclusion

Au cours de ce chapitre, nous avons présenté le problème de vérification temporelle et sa modélisation sous forme de graphe pour les systèmes acycliques. Nous avons aussi donné la formulation du problème de calcul de maximum de séparation pour les systèmes acycliques sous forme de problème d'optimisation et présenté les différentes notions et termes liés au problème de vérification qui seront nécessaires à la compréhension de ce mémoire. Dans le chapitre suivant, nous nous intéressons au problème de vérification temporelle pour les systèmes cycliques.

Chapitre III Vérification temporelle des systèmes cycliques

Au cours de ce chapitre, on va s'intéresser à l'étude du problème de vérification temporelle pour les systèmes cycliques, qui constitue le sujet principal de ce mémoire. Ce chapitre est consacré à la présentation du problème, le graphe d'événements adapté utilisé dans ce mémoire, la formulation sous forme de problème d'optimisation, la complexité et les solutions proposées par d'autres chercheurs pour le résoudre.

1 Adaptation du graphe d'événements et notion de hauteur

Les systèmes cycliques sont des systèmes numériques qui effectuent des traitements itératifs au cours du temps (ce qui est le cas de la plupart des systèmes numériques). Dans ces systèmes, les relations entre les événements peuvent être inter-itération ou intra-itération. Les relations intra-itération représentent des contraintes entre deux événements de la même itération. Cependant, les relations inter-itération représentent des contraintes entre deux événements appartenant à des itérations différentes. La notion de hauteur entre deux événements permet d'indiquer s'il s'agit d'une relation intra-itération (dans ce cas la valeur de la hauteur est nulle) ou bien s'il s'agit d'une relation inter-itération et dans ce cas la valeur de la hauteur indique le nombre d'itérations qui les séparent. Notons que jusqu'à présent les travaux que nous avons trouvés et qui traitaient des systèmes cycliques ont restreint la valeur de la hauteur à zéro ou à un, c'est-à-dire qu'ils se sont limités aux systèmes cycliques qui ont des événements qui se répètent à chaque itération ou qui ne se répètent pas (c'est souvent le cas des événements exécutés une seule fois lors de démarrage du système ou après un signal de remise à l'état initial).

Dans ce travail, nous avons étendu les graphes d'événements pour pouvoir modéliser et représenter le comportement itératif et inclure la notion de hauteur.

D'où la définition du graphe d'événements *G(E, A, L, U, H, T)* suivante:

- E= {e} représente l'ensemble des événements et constitue l'ensemble des nœuds du graphe d'événements.
- $A \subset E \times E$: représente l'ensemble des arcs du graphe d'événements qui relient toute paire d'événements ayant une contrainte temporelle entre eux. On note par a_{ij} l'arc de i vers j. À chaque arc $a_{ij,}$ on associe une étiquette ([l_{ij}, u_{ij}], h_{ij}) composée d'un intervalle avec une borne inferieure et une borne supérieure, notées respectivement l_{ij} et u_{ij}, représentant le délai entre les temps de déclenchement des événements i et j, et h_{ij} désigne la valeur de la hauteur.
- $L : A \rightarrow \mathbf{N}$ (l'ensemble des entiers naturels) : La valeur $L(a_{ij}) = l_{ij}$ représente le temps minimal du déclenchement d'un événement j par rapport à son prédécesseur i.
- $U : A \rightarrow \mathbf{N}$ (l'ensemble des entiers naturels) La valeur $U(a_{ij}) = u_{ij}$ représente le temps maximal du déclenchement d'un événement j par rapport à son prédécesseur i, avec $U(a_{ij}) \geq L(a_{ij})$ pour tout arc $a_{ij.}$
- $H : A \rightarrow \mathbf{N}$ est la fonction de la hauteur qui représente le nombre d'itérations qui séparent deux événements et définit si la relation entre eux est intra-itération ($h_{ij} = 0$) ou inter-itération ($h_{ij} > 0$ avec h_{ij} est la valeur de la hauteur entre i et j).
- $T : E \rightarrow$ {lin, min, max} : $T(e)$ désigne le type de contrainte associé à chaque événement. Le type de contrainte détermine la relation entre les temps d'occurrence d'un événement et ses prédécesseurs, comme nous l'avons décrit précédemment.

Après avoir présenté la notion de hauteur et le graphe utilisé dans ce travail pour la modélisation des systèmes cycliques, nous donnons la formulation du problème de maximum de séparation pour les systèmes cycliques, qui est similaire à celle des systèmes acycliques, sauf qu'on associe un indice aux événements pour indiquer le numéro de l'itération de l'événement.

Formulation du problème de maximum de séparation pour les systèmes cycliques

$$Max : t(k_c) - t(l_d)\,, \qquad k_c \; et \; l_d \; donnés$$

$$\begin{cases} t\left(j_{n+h_{ij}}\right) - \; t(i_n) \leq \; u_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = min \\ t(i_n) - t\left(j_{n+h_{ij}}\right) \leq \; -l_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = max \\ t\left(j_{n+h_{ij}}\right) \leq \max(t(i_n) + u_{ij}), i \in pred(j), \qquad T(j) = max \\ \min(t(i_n) + l_{ij}) \leq \; t\left(j_{n+h_{ij}}\right), i \in pred(j), \qquad T(j) = min \end{cases}$$

2 Complexité et travaux précédents

Théorème : Le problème de séparation maximum pour les systèmes cycliques avec des contraintes Min-Max-Lin est NP-complet

Preuve : On sait que le problème de maximum de séparation pour les systèmes acycliques est NP-complet. Tout système acyclique est un système cyclique ayant toutes les hauteurs égales à zéro. Donc le problème de maximum de séparation pour les systèmes cycliques avec des contraintes min-max-lin est NP-complet.

Il n'y a pas beaucoup de travaux qui ont traité le cas avec les différents types de contraintes pour les systèmes cycliques, la plupart proposent des solutions pour un sous-ensemble de systèmes. Beaucoup de solutions existantes se sont basées sur le dépliage. Mais l'utilisation de dépliage peut poser un problème au niveau du nombre de dépliages requis et la taille du problème (même si on utilise un algorithme polynomial comme pour les systèmes avec contraintes `max` uniquement, si la taille du graphe devient très grande, après le dépliage, le temps de résolution requis augmente considérablement). C'est pour cette raison que les chercheurs prennent des systèmes (graphes) particuliers avec des caractéristiques définies qui garantissent la convergence de ces systèmes après un nombre de dépliages fini. Notons que dans notre travail, nous n'utilisons pas le dépliage; de ce fait, nous n'allons pas décrire la procédure de dépliage surtout qu'elle est assez complexe pour les graphes avec des hauteurs non limitées à 0 ou 1. Pour des détails sur l'algorithme de dépliage, vous pouvez vous référer à [12].

Les travaux d'Amon *et al.* [13] sont un exemple de travaux basés sur le dépliage. Ils utilisent une technique algébrique pour le calcul du maximum du temps de séparation entre deux événements d'un graphe constitué uniquement de contraintes `max`. Leur technique est exponentielle en terme de la taille du problème [5].

Vu la complexité du problème surtout avec les contraintes min-max-linéaire, beaucoup de chercheurs ont opté pour des solutions approximatives pseudo-polynomiales, au lieu de chercher des algorithmes exacts avec des temps exponentiels au pire des cas. C'est dans ce cadre que se situe l'un des travaux les plus pertinents dans ce domaine, celui de Chakraborty *et al.* [5] qui ont proposé un algorithme approximatif pseudo-polynomial pour le calcul des temps de séparation des événements avec des contraintes `min` et `max` pour des systèmes asynchrones avec des bornes finies. Leur méthode est divisée en deux étapes.

La première étape est une étape d'analyse du comportement du système à long terme, qui consiste à calculer les temps de séparation des événements après que le système soit activé pendant une longue période de temps. Cette étape consiste à déplier le graphe successivement en générant des sous-graphes isomorphes, puis calculer les séparations jusqu'à atteindre le point de convergence *k,* qui désigne l'itération à partir de laquelle $\Delta_i(a,b) = \Delta_{i+1}(a,b)$, où $\Delta_i(a,b)$ représente le temps de séparation entre les événements *a* et *b* à l'itération *i*(chaque dépliage correspond à une itération). Cette phase s'arrête si le système a convergé ou bien s'il a atteint k_{max} (un nombre de dépliages spécifié à l'avance pour éviter de déplier à l'infini); dans ce cas, les résultats seront conservateurs. Les séparations calculées s'appliquent pour tous les sous-graphes G_i tel que $i \geq k$.

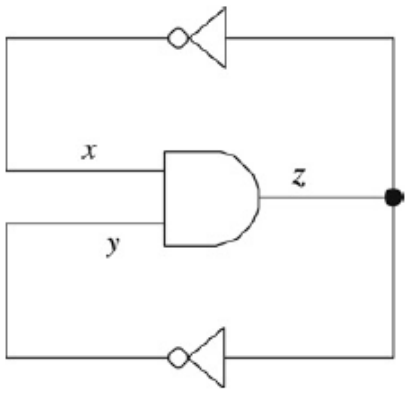


Figure 5 Exemple d'un simple circuit [5]

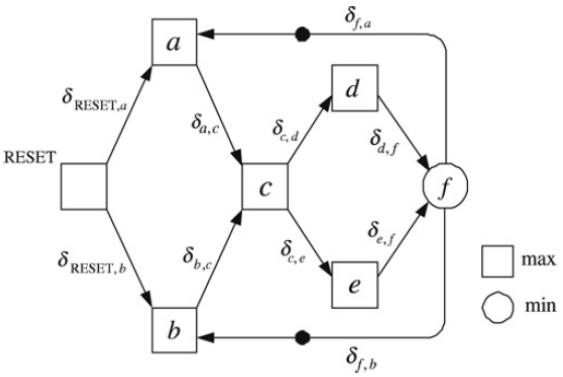


Figure 6 Le graphe cyclique correspondant au circuit de la Figure 5 avec des contraintes `min` et `max` [5], les arcs avec un point indiquent que la hauteur est égale à 1

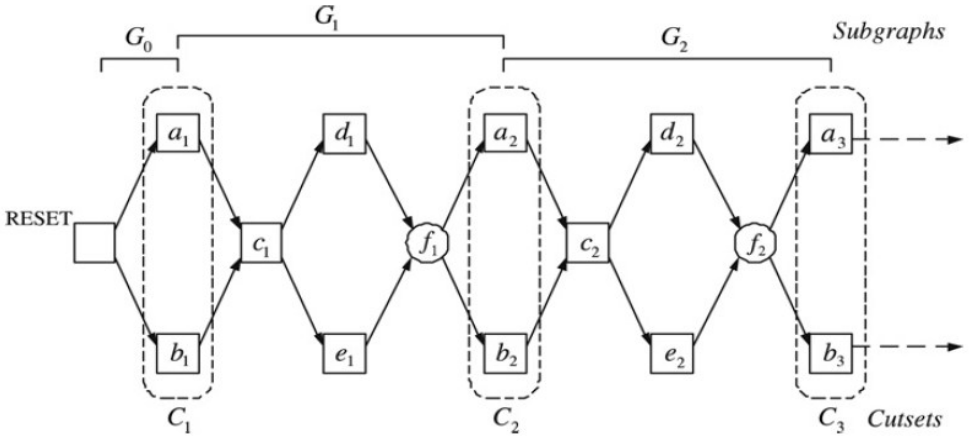


Figure 7 Dépliage du graphe de la Figure 6 montrant les sous-graphes G_i [5]

Dans la deuxième étape, ils étudient le comportement initial du système (dès le lancement ou après un signal *reset*). Cela consiste à construire un graphe acyclique à partir des sous-graphes dépliés, G_0 jusqu'à $G_{k\text{-}1}$ (k obtenu dans la première étape ou k_{max} si le système ne converge pas), puis à appliquer un algorithme de calcul de temps de séparation acyclique pour obtenir les temps de séparation.

Bien que cette approche ait une complexité pseudo-polynomiale $O((m+k_{max}n)^2p)$ (avec n le nombre totale des événements, m le nombre des nœuds sources (sans prédécesseurs) et p le nombre maximal de prédécesseurs d'un événement), elle souffre de plusieurs limites. D'abord, elle s'applique uniquement sur une catégorie de graphes bien particulière appelé « tightly coupled » (c'est un graphe avec certaines propriétés définies dans [6] et dépasse le propos de ce mémoire).

Cette limite aux systèmes « tightly coupled » restreint le champ d'application de cette approche à un type particulier de systèmes. Par exemple, l'une des propriétés exige d'avoir un nombre d'événements `max` beaucoup plus important que le nombre d'événements `min`. En plus, afin de garantir la convergence et l'exactitude des résultats, ils exigent que toutes les bornes des contraintes soient finies et connues à l'avance, ce qui est très restrictif vu que la plupart des systèmes contiennent des bornes infinies utilisées surtout pour modéliser les temps d'occurrences des événements produits par l'environnement qui sont inconnus à l'avance. Notons aussi, que même si toutes les propriétés sont vérifiées, les résultats restent toujours conservateurs [5].

Dernièrement en 2008, Zhao *et al.*[14] ont étendu et adapté l'algorithme CAS (CyclicApproxSep) de [5] pour couvrir une classe de systèmes plus large que les systèmes « tightly coupled » appelés systèmes uniformes (l'explication de ces systèmes est complexe et dépasse le propos de ce mémoire, pour plus de détails voir [14]). Ils ont défini les conditions nécessaires et suffisantes pour le calcul des temps de séparation pour les systèmes cycliques avec hauteur inférieure ou égale à un. En résumé, leur méthode est basée sur une analyse algébrique des squelettes de graphes afin de vérifier la condition d'uniformité puis calculer les temps de séparation à l'aide d'une version modifiée de l'algorithme CAS vu précédemment. L'avantage majeur de ce travail réside dans le fait de couvrir une classe plus large que les systèmes « tightly coupled », néanmoins la classe de systèmes couverte reste limitée. En plus, la hauteur est restreinte à zéro ou un et il faut encore perfectionner l'étape de test de l'uniformité [14].

3 Conclusion

Dans ce chapitre, nous avons présenté les systèmes cycliques, puis nous avons donné le graphe utilisé dans ce mémoire et la formulation du problème de maximum de séparation pour les systèmes cycliques sous forme de problème d'optimisation. Nous avons aussi présenté la complexité du problème et passé en revue quelques travaux précédents. Dans le chapitre suivant, nous présenterons notre solution pour la résolution du problème en le transformant sous forme d'un MILP afin de le résoudre par la suite à l'aide du solveur Cplex.

Chapitre IV Modélisation sous forme MILP

Au cours de ce chapitre, nous présentons notre solution pour résoudre le problème de maximum de séparation pour les systèmes cycliques cités précédemment. Nous présentons les différentes étapes de transformation du problème de maximum de séparation en un programme linéaire en nombres entiers mixtes (MILP), afin de le résoudre par la suite à l'aide du solveur Cplex. Tout d'abord, nous commençons par présenter l'étape de linéarisation des contraintes `min` et `max` du problème. Ensuite nous présentons la transformation des relations inter-itérations en relation intra-itérations. Finalement nous donnons la formulation du problème de maximum de séparation sous forme MILP et son équivalent avec la syntaxe du solveur Cplex. Nous terminons ce chapitre par la description de l'outil développé (format de fichiers et de données utilisé, les différentes méthodes et fonctionnalités) dans le cadre de ce mémoire afin de tester notre approche.

1 Linéarisation des contraintes min et max

D'abord, rappellons la formulation du problème de maximum de séparation pour les systèmes cycliques donnée dans le chapitre précédent et qui sera le point de départ pour ce chapitre :

$$Max : t(k_c) - t(l_d)\,, \qquad k_c \; et \; l_d \; donnés$$

$$\begin{cases} t\left(j_{n+h_{ij}}\right) - \; t(i_n) \leq \; u_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = min \\ t(i_n) - t\left(j_{n+h_{ij}}\right) \leq \; -l_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = max \\ t\left(j_{n+h_{ij}}\right) \leq \max(t(i_n) + u_{ij}), i \in pred(j), \qquad T(j) = max \\ \min(t(i_n) + l_{ij}) \leq \; t\left(j_{n+h_{ij}}\right), i \in pred(j), \qquad T(j) = min \end{cases}$$

La première étape de transformation du problème de maximum de séparation sous forme de MILP est de linéariser les fonctions `min` et `max`.

Afin de linéariser les fonctions min et max nous avons utilisé le procédé de linéarisation en utilisant une grande constante entière M (en anglais connu sous le nom« Big-M reformulation »). Ce procédé de linéarisation très connu dans le domaine de recherche opérationnelle et utilisé dans les travaux de [2] en 1994 et aussi dans [11] en 2009 pour résoudre le problème de maximum de séparation, consiste à transformer les fonctions min et max de la manière suivante.

Chaque fonction `max` de la forme $z = \max(x, y)$ est remplacée par les inéquations linéaires suivantes, où M est une très grande constante:

$$z = \max(x, y) \Leftrightarrow \begin{cases} z \; \geq \; x \\ z \; \geq y \\ z \leq x + M\alpha_i \\ z \leq y + M(1 - \alpha_i) \\ \alpha_i \in \{0,1\} \end{cases}$$

De façon similaire la fonction `min` est remplacée par les équations linéaires suivantes :

$$z = \min(x, y) \Leftrightarrow \begin{cases} z \; \leq \; x \\ z \; \leq y \\ z \geq x + M\alpha_i \\ z \geq y + M(1 - \alpha_i) \\ \alpha_i \in \{0,1\} \end{cases}$$

*Notons que *M* doit être au moins égale à la différence entre *x* et *y*

Pour résoudre ce système d'inéquations, il faut explorer deux solutions, celle avec $\alpha_i = 1$ et celle $\alpha_i = 0$, dans un cas on obtient la solution recherchée, c'est-à-dire le `min` ou le `max`, dans l'autre l'ensemble vide.

Exemple z= max(5,7),(en considérant M=10) on obtient :

Pour $\alpha_i = 0$

On obtient le système d'inéquations suivant:

$$\begin{cases} z \geq 5 \\ z \geq 7 \\ z \leq 5 \\ z \leq 17 \end{cases} \Rightarrow \begin{cases} z \geq 7 \\ z \leq 5 \end{cases}$$

On obtient l'ensemble vide

Pour $\alpha_i = 1$

On obtient le système d'inéquations suivant:

$$\begin{cases} z \geq 5 \\ z \geq 7 \\ z \leq 15 \\ z \leq 7 \end{cases} \Rightarrow \begin{cases} z \geq 7 \\ z \leq 7 \end{cases}$$

On obtient z=7 = max(5,7)

En appliquant cette transformation, toutes les inéquations de la formulation du problème de maximum de séparation deviennent linéaires en fonction des variables originales augmentées des nouvelles variables α_i. L'utilisation des variables binaires peut résulter en une exécution exponentielle en fonction du nombre de contraintes min-max transformées (au pire des cas il va falloir explorer toutes les combinaisons possibles des variables binaires), ce qui explique l'utilisation d'heuristiques basées sur la méthode du Branch & Bound dans [2] et [11]. Car si nous avons *m* variables binaires, alors il se peut qu'on ait à explorer 2^m solutions, étant donné que chaque variable binaire admet deux solutions à explorer pour chaque contrainte `min` ou `max`. Au pire des cas, on aura 2^m explorations à faire avec *m* le nombre de contraintes min et max, d'où la nécessité d'une heuristique permettant de réduire le nombre d'explorations afin de déterminer les valeurs des variables binaires en évitant le parcours exhaustif de toutes les combinaisons possibles. Pour résoudre ce problème on a utilisé le solveur Cplex qui est l'un des plus performants actuellement. Sauf qu'on s'est confronté à un deuxième problème posé par la linéarisation, relié au choix de la constante *M,* car elle doit être la plus proche possible de la différence entre *x* et *y* sinon on aura des valeurs erronés et des contraintes violées (d'ailleurs on a noté ce problème aussi dans le travail de [11] qui donne quelques valeurs erronées dont le nombre augmente avec la taille du graphe). En effet, l'utilisation de grand entier *M* choisie au hasard (choix de grandes valeurs aléatoires de l'ordre de 10^6 et plus) a de graves conséquences sur le comportement numérique du solveur d'une part et sur le temps de calcul d'une autre part.

Pour mieux illustrer les problèmes causés par l'utilisation de très grand entier *M* aléatoires on va utiliser un exemple :

Soit :

$$(1)\ x - M\alpha \leq 0, avec\ M = 10^9\ et\ \alpha \in \{0,1\}$$

Par relaxation, $\alpha \in \{0,1\}$ devient $\alpha \in [0,1]$. Cette inéquation peut être satisfaite par la solution « quasi-entière » (x=10, α=10^{-8}), en effet 10^{-8}est tellement petit que le solveur va le considérer comme étant une valeur entière nulle. Or, en prenant α = 0 on obtient : « $10 \leq 0$ », ceci explique les valeurs erronées et les contraintes violées que nous avons obtenues lors de nos premières expériences. En plus ces erreurs ne sont pas détectées par le solveur Cplex et peuvent se propager et causer d'autres erreurs, ceci est détaillé dans l'exemple suivant.

Le deuxième problème est causé par l'utilisation d'un mélange de petits et grands entiers, ceci a de graves conséquences sur la décomposition LU utilisée par le solveur [15]. Supposons qu'on a une deuxième inéquation :

$$(2)\ 0.33333z - \alpha = 0$$

Si au cours de la factorisation on décide d'éliminer α. Alors il faut multiplier (2) par 10^9 et la soustraire de (1), ce qui donnerait :

$$(1) - 10^9(2)\ :\ x - 333330000\mathrm{z} \leq 0$$

En fait, l'utilisation de petites valeurs imprécises telles que 0.33333 (qui probablement aurait dû être 1/3), et les erreurs d'arrondissement dans les calculs précédents, causera la propagation des erreurs si la décomposition utilise de grands poids pour éliminer les non-zéro (tel que 0.33333).

La meilleure solution qu'on a trouvée pour résoudre ces problèmes est l'utilisation d'indicateurs de contraintes proposés par le solveur Cplex depuis sa version 10 (dans ce mémoire, on utilise la version 12). Les indicateurs de contraintes sont un moyen pour exprimer des relations entre les variables en identifiant une variable binaire pour contrôler si oui ou non une contrainte linéaire spécifiée est active [16]. Selon la documentation de Cplex, les indicateurs de contraintes ont été introduits spécialement pour résoudre ce type de problème et remplacer la reformulation de grand entier *M*. Le premier problème est résolu par le fait que les indicateurs de contraintes sont

traités par le branchement, dans l'exemple de l'inéquation (1) après le branchement sur α=0, la variable x est inférieure ou égale à 0. Le deuxième problème est résolu par le fait que les indicateurs de contraintes (les variables binaires) ne font pas partie de la matrice de coefficients et donc n'influent pas la décomposition LU.

Revenons maintenant à nos contraintes min et max, comme expliqué précédemment dans le chapitre 2 une seule borne pose un problème pour ces deux types de contraintes : la borne supérieure pour les contraintes max et la borne inférieure pour les contraintes min, en utilisant les indicateurs de contraintes:

La borne supérieure des contraintes max

$$t\left(j_{n+h_{ij}}\right) \leq \max(t(i_n) + u_{ij}), i \in pred(j), \qquad T(j) = max$$

est remplacée par:

$$\begin{cases} \alpha_i = 1 \Rightarrow t\left(j_{n+h_{ij}}\right) \leq t(i_n) + u_{ij}, i \in pred(j), \qquad T(j) = max \\ \sum_{i \in pred(j)} \alpha_i = 1, \ \ \alpha_i \in \{0,1\}^* \end{cases}$$

la borne inférieure des contraintes min :

$$\min(t(i_n) + l_{ij}) \leq \ t\left(j_{n+h_{ij}}\right), i \in pred(j), \qquad T(j) = min$$

est remplacée par :

$$\alpha_i = 1 \Rightarrow t(i_n) + l_{ij} \leq t\left(j_{n+h_{ij}}\right), i \in pred(j), \qquad T(j) = min$$
$$\sum_{i \in pred(j)} \alpha_i = 1, \ \ \alpha_i \in \{0,1\}^*$$

* $\sum_{i \in pred(j)} \alpha_i = 1, \ \ \alpha_i \in \{0,1\}$ reflète le fait qu'une seule borne est sélectionnée : la borne supérieure maximale pour le type max et la borne inférieure minimale pour le type min comme expliqué précédemment.

En utilisant les indicateurs de contraintes pour modéliser les contraintes min et max, nous avons :

- Transformé toutes les contraintes du problème de maximum de séparation en contraintes linéaires;
- Réussi à refléter le fait qu'une seule borne est sélectionnée pour ces types de contraintes (uniquement la borne supérieure maximale pour le type `max` et la borne inférieure minimale pour le type `min` comme expliqué précédemment dans le chapitre 2) à l'aide des variables binaires;
- Réussi à obtenir des résultats exacts et corrects en évitant les problèmes liés à l'utilisation de grand *M*.

Après la linéarisation des contraintes min et max en utilisant les indicateurs de contraintes, on obtient la formulation du problème de maximum de séparation suivante :

$$Max : t(k_c) - t(l_d)\,, \qquad k_c \; et \; l_d \; donnés$$

$$\begin{cases} t\left(j_{n+h_{ij}}\right) - \; t(i_n) \leq \; u_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = min \\ t(i_n) - t\left(j_{n+h_{ij}}\right) \leq \; -l_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = max \\ \begin{cases} \alpha_i = 1 \Rightarrow t\left(j_{n+h_{ij}}\right) \leq t(i_n) + u_{ij}, i \in pred(j), \qquad T(j) = max \\ \sum_{i \in pred(j)} \alpha_i = 1, \;\; \alpha_i \in \{0,1\} \end{cases} \\ \begin{cases} \alpha_i = 1 \Rightarrow t(i_n) + l_{ij} \leq t\left(j_{n+h_{ij}}\right), i \in pred(j), \qquad T(j) = min \\ \sum_{i \in pred(j)} \alpha_i = 1, \;\; \alpha_i \in \{0,1\} \end{cases} \end{cases}$$

2 Transformation des relations inter-itérations en relation intra-itérations

La deuxième étape de transformation du problème de maximum de séparation en MILP consiste à transformer les relations inter-itérations par des relations intra-itérations. Pour ce faire, nous nous sommes inspirés du travail de [6] pour le problème d'ordonnancement cyclique. En fait, en linéarisant les contraintes min et max, nous avons obtenu un système d'inéquations semblable à

celui du problème d'ordonnancement. Par analogie, avec le problème d'ordonnancement, on observe qu'à partir d'un moment donné, il apparait un gabarit (sous graphe) qui se répète de manière périodique et identique dans le temps. Un exemple est donné dans les figures suivantes :

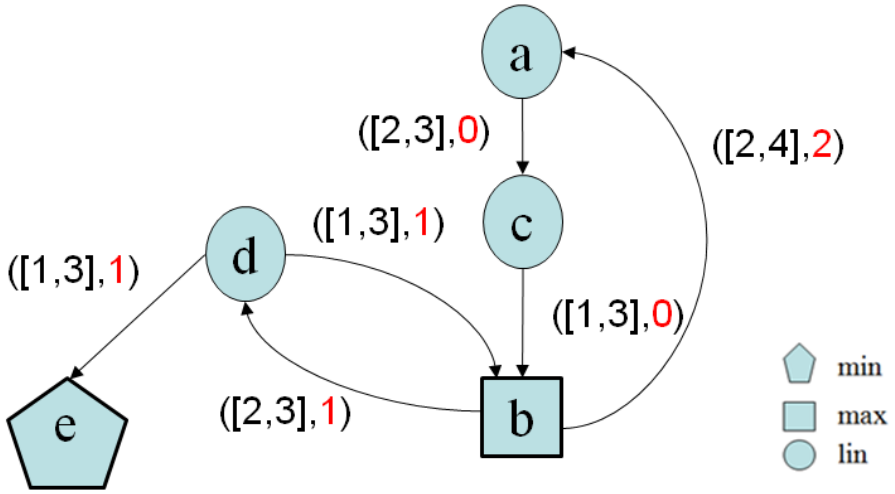


Figure 8 Exemple de graphe d'événement cyclique

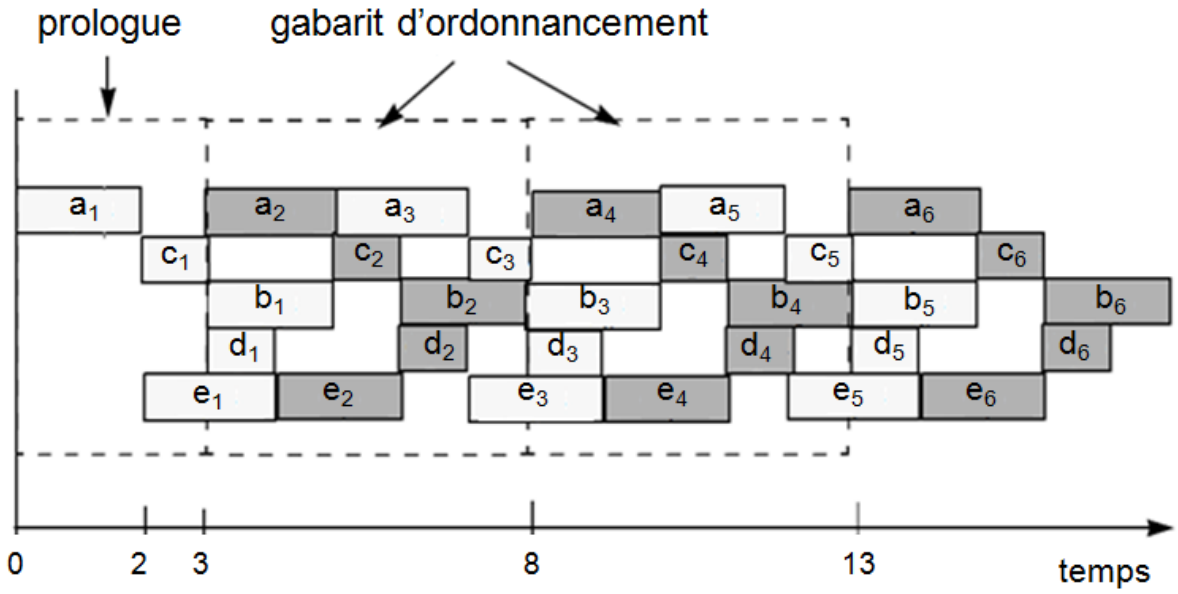


Figure 9 Ordonnancement correspondant au graphe de la Figure 8 avec période p=5/2

On note aussi au démarrage du système ou après un signal *reset*, la présence d'une partie qui précède la stabilisation du système et l'apparition du premier gabarit, cette partie correspond au prologue dans le problème d'ordonnancement [6]. Des systèmes avec un tel comportement sont

dits stables [6]. Les systèmes cycliques stables sont entièrement définis par le prologue, le gabarit et la fréquence qui désigne le nombre d'itération moyen par unité de temps (bien que pour notre problème on s'intéresse plus à la période, notée *p,* qui est égale à l'inverse de la fréquence).

En effet, dans le problème d'ordonnancement, on distingue deux types d'ordonnancement :

1. Un ordonnancement périodique dans lequel toutes les itérations ont le même ordonnancement, et la même durée sépare le début de deux itérations successives [6]. La fréquence correspondante $f= 1/P$ pour ce type d'ordonnancement avec P un rationnel égal à la durée d'une itération.
2. Un ordonnancement K-périodique dans lequel l'ordonnancement de K itérations successives est fixe et se répète à un intervalle régulier [6]. La fréquence correspondante $f = K/P$ avec K le nombre d'itérations qui compose le gabarit et P la durée de ces itérations. Notons que l'ordonnancement périodique est en effet un ordonnancement K périodique avec K=1.

Notons que, même si le système est constitué uniquement de contraintes linéaires et qu'il n y a pas de variables binaires, la recherche de la période p est pseudo-polynomiale [6].

Dans notre problème de maximum de séparation, la modélisation et la transformation des relations inter-itérations en relation intra-itérations sont effectuées comme suit :

Soit p un rationnel qui désigne la durée d'une itération; si on a une relation inter-itération entre deux éventements i et j et la hauteur h_{ij} désigne le nombre d'itérations entre ces deux évènements, alors le temps entre le déclenchement de l'événement i et celui de j est égal à $h_{ij} \times p$. D'où la nouvelle formulation du problème de maximum de séparation suivante:

$$Max : t(k_c) - t(l_d)\,, \qquad k_c \; et \; l_d \; donnés$$

$$\begin{cases} t(j) - t(i) + h_{ij} \times p \leq u_{ij}, i \in pred(j), T(j) = lin \; ou \; T(j) = min \\ t(i) - t(j) \leq -l_{ij} + h_{ij} \times p, i \in pred(j), T(j) = lin \; ou \; T(j) = max \\ \begin{cases} \alpha_i = 1 \Rightarrow t(j) - t(i) \leq u_{ij} - h_{ij} \times p, i \in pred(j), \qquad T(j) = max \\ \sum_{i \in pred(j)} \alpha_i = 1, \; \alpha_i \in \{0,1\} \end{cases} \\ \begin{cases} \alpha_i = 1 \Rightarrow t(i) - t(j) \leq l_{ij} + h_{ij} \times p, i \in pred(j), \qquad T(j) = min \\ \sum_{i \in pred(j)} \alpha_i = 1, \; \alpha_i \in \{0,1\} \end{cases} \end{cases}$$

Notons que dans le cas où le système n'admet pas un gabarit et aucun modèle n'apparaît ou ne se répète au cours du temps alors il nous sera impossible de le formuler de cette manière et par conséquent, de tels systèmes ne peuvent pas être analysés par la méthode proposée dans ce mémoire (on ne trouve aucun *p* qui vérifie le système d'inéquations ci-haut).

Étant donné qu'on s'est inspiré du problème d'ordonnancement pour résoudre le problème de calcul des maximums des temps de séparation, la résolution nous a permis d'avoir les temps de séparation et par la même occasion l'ordonnancement correspondant à la période *p*.

Bien que, le problème d'ordonnancement et celui de vérification temporelle aient une formulation similaire, leurs objectifs sont différents. Dans le problème d'ordonnancement, on cherche à l'optimiser de manière à avoir le plus d'opérations en moins de temps possible et donc réduire *p* le plus possible (on cherche à minimiser *p*). Par contre, pour le problème de vérification temporelle on cherche le temps maximum qui sépare les événements.

3 Outil développé

Au cours de ce mémoire, nous avons développé un outil pour calculer le temps de séparation maximum pour des graphes cycliques et acycliques.

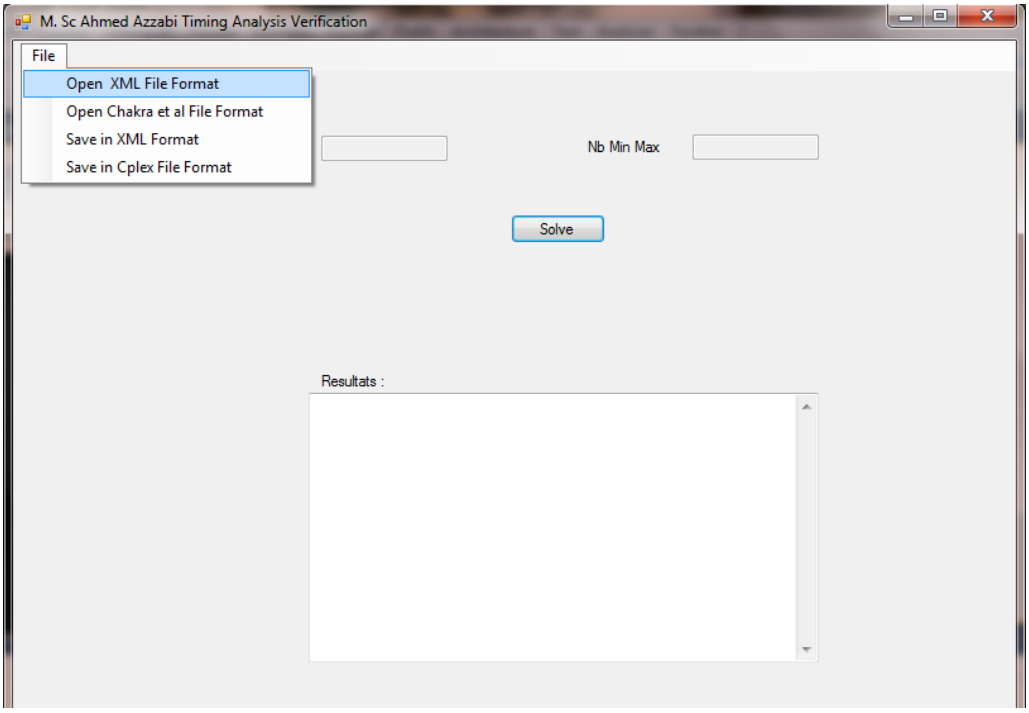


Figure 10 Menu principal de l'outil développé

L'outil est développé avec Visual Studio 2008 en utilisant le langage C#. Pour Cplex, nous avons utilisé la version 12 via une libraire .Net.

Le solveur Cplex est l'un des solveurs les plus performants existant actuellement. Il utilise des algorithmes optimisés pour la résolution de problèmes de grande taille tout en minimisant le temps de résolution. Entre autre, il permet de résoudre différents types de programmes mathématiques : programme linéaire, programme linéaire d'entiers mixtes, programme quadratique et quadratique d'entier mixte. Voir [16] pour plus de détails sur le solveur Cplex.

3.1 Fonctionnement général de l'outil

En entrée, notre application accepte des graphes d'événements enregistrés dans des fichiers sous format XML (détaillé plus loin). Après la lecture du fichier, nous faisons un « parsing » du fichier passé en entrée et nous remplissons une structure de graphes internes afin de pouvoir effectuer les traitements. Ensuite, l'application génère le modèle Cplex correspondant à partir du graphe. Finalement, on lance le calcul des temps de séparation entre les paires d'événements, puis le résultat est donné sous forme de matrice qui contient les temps de séparation entre les

événements. Notre application permet aussi d'accepter le format de fichier de [5] afin qu'on puisse effectuer les expériences sur la puce DiffEq. Entre autre, l'application permet de générer des graphes sous format de fichier XML et elle génère aussi un fichier Cplex qui peut être utilisé directement avec la console interactive Cplex pour des fins de débogage ou paramétrage personnalisé. L'application est résumée par le schéma suivant :

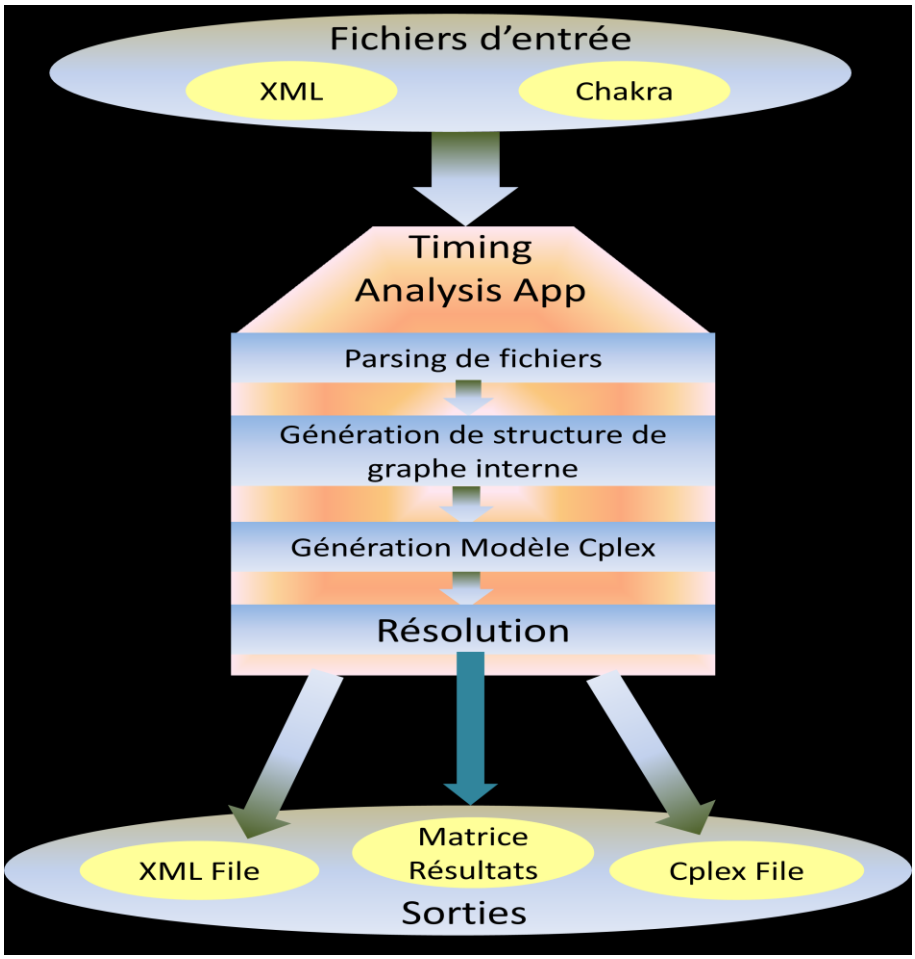


Figure 11 Schéma résumant l'application « Timing Analysis App »

Remarques :

- L'application contient d'autres fonctionnalités détaillées dans TABLEAU 2.
- L'application peut accepter n'importe quel type de fichier, il suffit juste d'ajouter le parseur adéquat pour récupérer les données du graphe.

3.2 Format de données

Pour la représentation des graphes, nous avons utilisé le format de fichier XML vu sa structure intuitive et très adapté pour la représentation des graphes ainsi que les nombreux avantages qu'ils offrent :

- La lisibilité : le format XML est intuitif et facile à comprendre sans avoir besoin de connaissance préalable.
- Une structure arborescente : ce qui permet de modéliser la majorité des problèmes informatiques.
- Intégrabilité et portabilité : vu que c'est un format universel et que de nombreux langages de programmation et outils offre des API de gestion, parsing et échange de fichier XML.
- Extensible : permet de décrire son propre langage (grammaire, vocabulaire).

Le véritable contenu d'un document XML est représenté sous forme d'un arbre d'éléments. Ces élément sont contenus dans des balises et peuvent éventuellement avoir des attributs de la frome suivante : **<élément attribut="valeur"/>.** Pour plus de détails sur XML référez-vous à [17].

Dans ce travail on a utilisé trois éléments :

- **<graph> :** représente le nœud principal de l'arbre XML et n'a pas d'attribut.
- **<event> :** représente un événement et il est caractérisé par deux attributs : **"label"** et **"type"** qui désigne respectivement l'étiquette de l'événement et le type de l'événement (min, max ou lin).
  Exemple : Un événement *i* de type max : <event label="i" type="max" />.
- **<edge>** : représente un arc du graphe et il est caractérisé par cinq attribut "**source**" qui désigne le nœud source de l'arc, "**destination**" qui désigne le nœud destination de l'arc, "**lower**" désigne la borne inférieure, "**upper**" désigne la borne supérieure et "**height**" désigne la valeur de la hauteur.
  Exemple : <**edge source**="i" **destination**="j" **lower**="2" **upper**="2" **height**="1" />

Remarque : pour faciliter l'écriture des graphes acycliques, l'attribut hauteur est optionnel s'il n'est pas spécifié alors la hauteur est considéré par défaut égale à zéro.

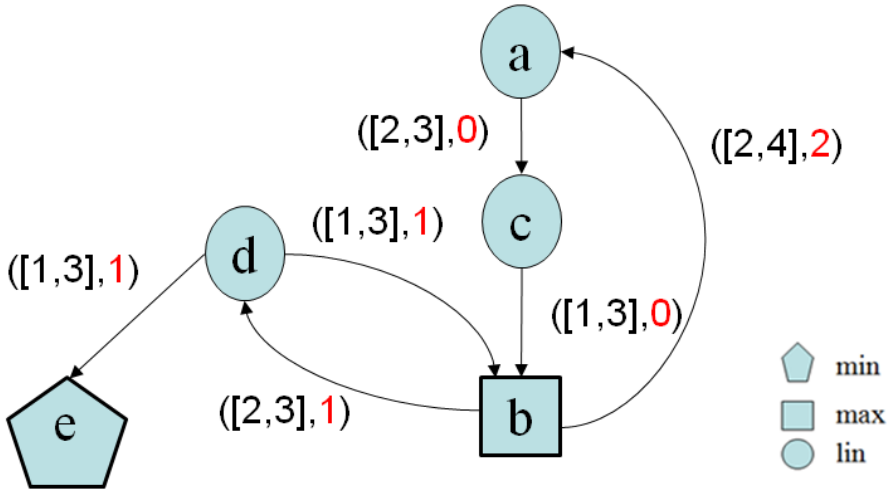


Figure 12 Exemple de graphe cyclique

```
<?xml version="1.0" ?>

<graph>
        <event label="a" type="lin" />
        <event label="b" type="max" />
        <event label="c" type="lin" />
        <event label="d" type="lin" />
        <event label="e" type="min" />
        <event label="f" type="lin" />
        <edge source="a" destination="c" lower="2" upper="3" height="0" />
        <edge source="c" destination="b" lower="1" upper="3" height="0" />
        <edge source="b" destination="a" lower="2" upper="4" height="2" />
        <edge source="b" destination="d" lower="2" upper="3" height="1" />
        <edge source="d" destination="b" lower="1" upper="3" height="1" />
        <edge source="d" destination="e" lower="1" upper="3" height="1" />
</graph>
```

Figure 13 Représentation XML du graphe de Figure 12

Le format XML a été utilisé pour représenter et sauvegarder les graphes dans des fichiers mais il n'est pas adéquat pour effectuer des traitements et manipuler le graphe. Alors, nous avons créé une structure de graphe offrant différentes fonctionnalités et permettant la manipulation des données.

Nous avons créé une classe "Graph" qui encapsule les différents composants du graphe d'événements utilisés dans ce mémoire ainsi que les différentes fonctionnalités qui lui sont liées.

La structure de graphe utilisé est un graphe de précédence vu la nature des traitements et la formulation du problème de maximum de séparation qui utilise souvent les prédécesseurs. La structure de graphes est donnée par la figure suivante :

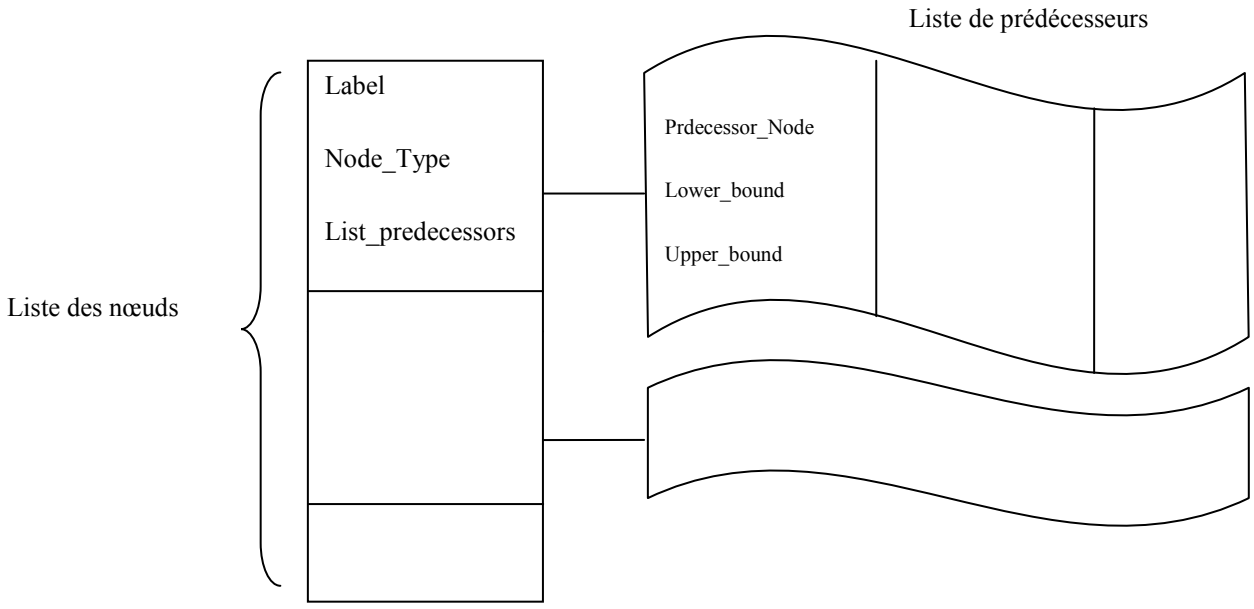


Figure 14 Structure de graphe de précédence utilisé dans ce mémoire

La liste des nœuds est implémentée sous forme d'un tableau dynamique. Chaque élément de ce tableau encapsule l'étiquette, le type de nœud (min, max, lin), et une référence sur la liste de ses prédécesseurs. La liste des prédécesseurs est implémentée sous forme de liste chainée dynamique, chaque élément de la liste est une structure qui encapsule le nœud prédécesseurs (indice dans le tableau de liste des nœuds pour un accès rapide), la borne inférieure et supérieure et la valeur de la hauteur.

Pour résoudre le problème de maximum de séparation, un modèle Cplex est généré à partir de la structure interne du graphe puis on fait appel à la méthode « Solve » du solveur pour avoir les maximum temps de séparation. La figure suivante illustre le modèle Cplex produit pour le graphe de la Figure 12 :

\Problem name: Maximum Separation

Max

\ Fonction objective: max sep entre a et b

Obj:a-b

Subject To

a-b+2p <=4

a-b+2p >=2

c-a<=3

c-a >=2

b-c<= 3

b-d +p <= 3

b-c>= 1

b-d +p >=1

alph =1 **->**d-b+p<= 3

alph =0 **->**d-b+p>= 1

e-d+p<= 3

e-d+p>= 1

\ Déclaration des variables entières

General

a b c d e

\ Déclaration des variables binaires

Binaries

alph

Figure 15 Format du modèle Cplex correspondant au graphe de la Figure 12

Les fonctionnalités offertes par la classe « Graph » sont résumées dans le tableau suivant :

Méthode	**Description**
ReadFromXML (File)	Lit à partir d'un fichier XML et remplit la structure de graphe
WriteToXML(File)	Écrit la structure de graphe dans un fichier XML
ReadFromCplex	Lit un fichier de format Cplex.
WriteToCplex	Écrit la structure de graphe dans un fichier Cplex
ReadFromChakra	Lit un fichier dans le format utilisé dans le travail de [5]et remplit la structure de graphe
FoldGraph	Replie les graphes de [5](Utilisé spécialement pour la DiffEqsolver)
Graph SuccessorGraph()	Convertit la structure de graphe de précédence en graphe de successeurs.
Max_Sep(node_i, node_j)	Calcule le maximum de séparation entre deux nœuds passés en paramètre
Solve()	Calcule le maximum de séparation entre les différentes paires de nœuds
Max_P()	Cherche la valeur maximale de la période *p*
Min_P()	Cherche la valeur minimale de la période *p*
LongestPath_Bellman_Ford(node_i, node_j)	Donne le plus long chemin entre deux nœuds sans utilisation du solveur (uniquement si le graphe est composé de nœuds de type linéaire)
Find_K_period()	Cherche *p* par la méthode décrite dans [6] pour les graphes linéaire uniquement
WriteResults()	Écrit les résultats (maximum de séparation) dans un fichier

TABLEAU 2 LISTE DES FONCTIONS DE L'APPLICATION

4 Conclusion

Au cours de ce chapitre, nous avons présenté notre démarche pour résoudre le problème de maximum de séparation en le formulant sous forme de MILP, puis le résoudre par le solveur

Cplex. Aussi nous avons décrit l'outil développé au cours de ce travail afin de pouvoir effectuer les tests et expérimentations sur différents systèmes cycliques et acycliques. Ces expérimentations feront l'objet du chapitre suivant dans lequel nous avons présenté toutes les étapes de l'approche analytique de la vérification temporelle, à travers l'exemple de la puce d'équation différentielle, en commençant par la modélisation graphique des composants sous forme de graphes à partir de la description RTL, puis la formulation des exigences sous forme de contraintes et finalement le calcul des temps de séparation et validation.

Chapitre V Expérimentation et résultats

Afin de tester notre approche, nous avons commencé par l'appliquer sur des exemples connus dans la littérature, le but était surtout de vérifier l'exactitude des résultats. Puis pour tester les performances, nous avons généré des grands graphes acycliques et cycliques d'une manière aléatoire détaillée plus loin dans ce chapitre. Et finalement, pour comparer et montrer l'efficacité de notre approche sur un cas réel non trivial, nous avons repris l'exemple d'une puce asynchrone avec traitement itératif de calcul d'équations différentielles conçue par Intel. Dans cet exemple nous allons détaillé tout le processus de l'approche analytique des contraintes temporelles, en commençant par la modélisation des composants sous forme de graphes puis la formulation des exigences sous forme des temps de séparations et en terminant par la détection et la localisation des violations et des erreurs.

1 Systèmes acycliques

Nous commençons par un exemple connu représentant un cycle de lecture d'une mémoire ROM reliée à un processeur Intel 8086 extrait de [11] et illustré par la Figure 16. Le système est constitué des composants suivants :

- Horloge émettant un signal chaque 204 ns (modélise par l'arc $c_1 \rightarrow c_2$ Figure 17)
- Loquet qui contient l'adresse [0,12ns] (modélise par l'arc $a_1 \rightarrow A_1$ Figure 17)
- Décodeur d'adresse [0,30ns] garantit que seulement la PROM sélectionnée met les données sur le bus à tout moment ($r1 \rightarrow A_1$ Figure 17).

Nous cherchons à vérifier si la PROM 2716 est assez rapide pour fonctionner avec le CPU8086 ayant une horloge de période 204ns.

En termes des temps de séparations, ceci revient à vérifier si :

$d_2 - d_1 \geq 0$; $d_2 - c_3 \geq 10$

$R_2 - R_1 \geq 0$; $A_2 - A_1 \geq 0$

$d_1 - a_2 \geq 0$; $c_3 - d_1 \geq 30$

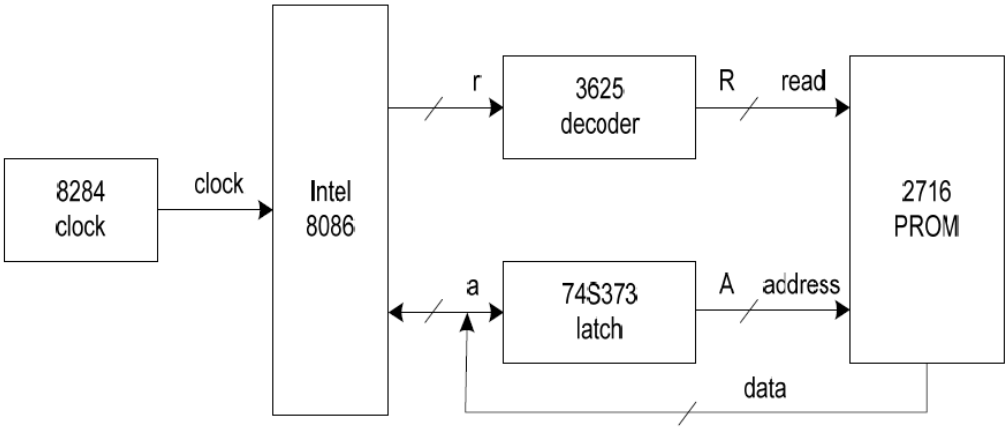


Figure 16 Cycle lecture Intel 8086-PROM [11]

Le graphe d'événements et contraintes correspondant est illustré par la Figure 17(puisqu'il est acyclique, la hauteur n'est pas indiquée). Le graphe est constitué de treize événements dont la signification est la suivante :

- c_1, c_2, c_3 transitions d'horloge
- a_1,a_2 : début et fin d'adresse valide du bus data/address.
- A_1, A_2 : désignent respectivement le début et la fin de l'entrée d'adresse valide dans le loquet.
- d_1: nœud de type max qui représente le début de données valide sur le bus qui se déclenche après que les deux signaux d'entrée ont eu lieu («address » ou « read signal »).
- d_2: nœud de type `min`, il représente la fin de données valides sur le bus qui se déclenche dès qu'un des deux signaux d'entrées est mis à zéro (address ou read signal).
- r_1/R_1, r_2/R_2: sont le début et fin de lecture de signaux address/read de la sortie de décodeur d'adresse.

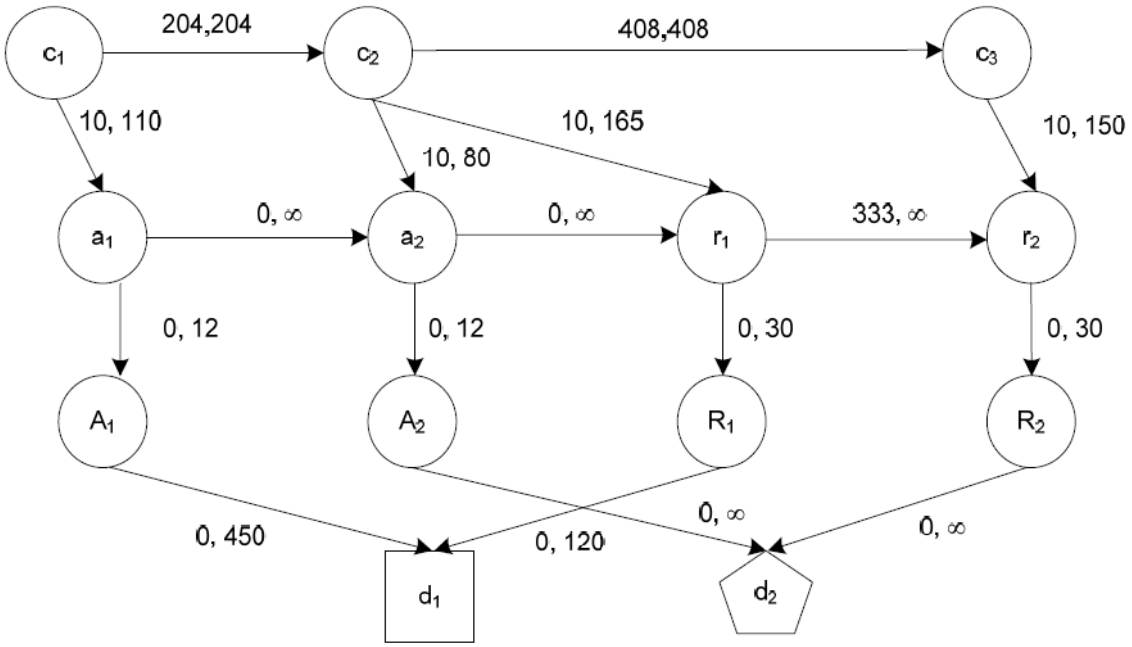


Figure 17 Graphe correspondant au Cycle de lecture de PROM-Intel 8086 [11]

Après transformation du graphe sous forme d'inéquations et le calcul des temps de séparation entre chaque paire d'événement à l'aide du solveur Cplex, on a obtenu les résultats illustrés dans le TABLEAU 3.

TABLEAU 3 TEMPS DE SEPARATIONS DU CYCLE DE LECTURE PROM-INTEL 8086 [11]

	c1	**c2**	**c3**	**a1**	**a2**	**r1**	**r2**	**A1**	**A2**	**R1**	**R2**	**d1**	**d2**
c1	0	204	612	110	284	369	762	122	296	399	792	572	∞
c2	-204	0	408	-94	80	165	558	-82	92	195	588	368	∞
c3	-612	-408	0	-502	-328	-243	150	-490	-316	-213	180	-40	∞
a1	-10	194	602	0	274	359	752	12	286	389	782	509	∞
a2	-214	-10	398	-104	0	155	548	-92	12	185	578	358	∞
r1	-214	-10	398	-104	0	0	548	-92	12	30	578	358	∞
r2	-622	-418	-10	-512	-338	-333	0	-500	-326	-303	30	-50	∞
A1	-10	194	602	0	274	359	752	0	286	389	782	509	∞
A2	-214	-10	398	-104	0	155	548	-92	0	185	578	358	∞
R1	-214	-10	398	-104	0	0	548	-92	12	0	578	358	∞
R2	-622	-418	-10	-512	-338	-333	0	-500	-326	-303	0	-50	∞
d1	-214	-10	398	-104	0	0	548	-92	12	0	578	0	∞
d2	-214	-10	398	-104	0	155	548	-92	12	185	578	358	0

Après la comparaison des résultats obtenus avec ceux requis, on remarque que plusieurs contraintes sont violées entre d_1 et d_2, et c_3 et d_2, ce qui permet de conclure que la PROM 2716 n'est pas assez rapide pour fonctionner avec le CPU 8086.

TABLEAU 4 COMPARAISON ENTRE LES SÉPARATIONS OBTENUES ET REQUISES

Contraintes	Intervalle de temps requis	Temps de séparation obtenu
$d2 - d_1$	[0, ∞]	**[-358, ∞]***
d_2-c_3	[10, ∞]	**[-398, ∞]***
R_2-R_1	[0, ∞]	[303, 578]
A_2-A_1	[0, ∞]	[92, 286]
d_1-a_2	[0, ∞]	[0, 358]
c_3-d_1	[30, ∞]	[40, 398]

*contraintes violées

2 Système cyclique

Pour les systèmes cycliques, nous avons repris l'exemple de [5]représenté par la Figure 18, malgré que la hauteur soit limitée à zéro ou un (les arcs marqués par un point dans la figure), car on n'a pas trouvé des travaux précédents qui ont une hauteur supérieure à 1et notre objectif était de vérifier l'exactitude de nos résultats comparés à d'autres approches.

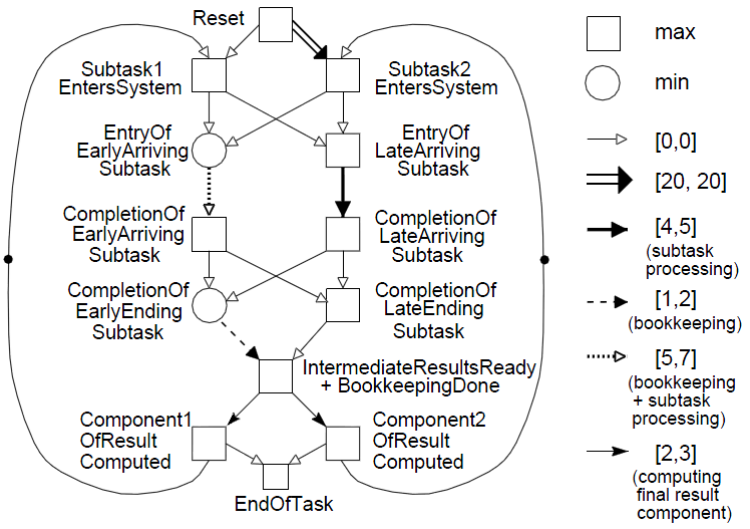


Figure 18Exécution des tâches par deux processeurs[5]. (Les arcs avec un point sont de hauteur un)

L'exemple représente deux processeurs qui exécutent des opérations d'une pile d'exécution. Chaque opération est divisée en deux sous-opérations, qui sont traitées par les deux processeurs en parallèle. Après le traitement, des résultats intermédiaires sont produits puis combinés ensemble pour donner le résultat final. Le calcul du résultat final se fait par les deux unités de résultats (désigné par « Component Of Result » dans la Figure 18) puis le résultat est envoyé au gestionnaire des tâches qui va charger la prochaine opération dans l'un des processeurs inactif sans attendre que l'unité de résultat termine, de cette manière plusieurs tâches (opérations) sont exécutées en parallèle. Ces opérations sont supposées se répéter à l'infini, pour plus détails voir [5]. Le but était de déterminer les délais d'exécution d'une opération (minimum et maximum temps d'exécution d'une opération). Après le calcul de temps de séparation entre le début et la fin d'exécution d'une tâche nous avons obtenu les mêmes résultats que [5], c'est-à-dire un délai situé entre [7,11] cycle d'horloge après la stabilisation du système, puisque lors de la phase du démarrage, il y a un délai d'initialisation égal à 20 (Figure 18, arc entre Reset et SubTask2).
Mis à part cet exemple, nous avons repris deux autres petits exemples dont celui de Amon *et al.*[6] et nous avons eu les mêmes temps de séparation, ce qui confirme "« l'exactitude »de notre approche et notre modélisation sous forme de MILP.

Vu l'absence des travaux traitant les systèmes cycliques de hauteur supérieure à un, ainsi que le manque d'exemples non triviaux pour les systèmes acycliques, nous avons généré des graphes aléatoires afin de tester les performances de notre approche et la comparer avec d'autres approches. Le TABLEAU 6donne quelques résultats.

La difficulté liée à la génération de graphes aléatoires réside dans le fait qu'ils doivent être consistants, pour ce faire, on regroupe différents petits graphes consistants ensemble, ayant chacun un nœud source et un nœud final, puis on les connecte les uns aux autres pour former un plus grand graphe consistant.

3 Application

Afin de mettre en pratique notre approche et voir son efficacité, nous avons repris l'exemple de [5], d'un système cyclique réel d'une puce asynchrone de résolution d'équations différentielles. On a réussi à obtenir les mêmes valeurs de temps de séparations exactes et sans faire de dépliage (contrairement aux travaux précédents). Nous avons pu vérifier les différentes contraintes

temporelles de la puce et nous avons découvert un bug en quelques secondes que le simulateur SPICE et d'autres méthodes exactes ont mis des heures à découvrir.

Dans cette section, nous allons commencer par décrire la puce DiffEq et la procédure utilisée afin de la modéliser et passer d'une description RTL vers un graphe d'événements et de contraintes. Ensuite, nous décrirons comment les contraintes temporelles sont vérifiées, comment déterminer s'il y a des violations ou non et si la puce répond aux exigences. La description et la modélisation sont adaptées et extraites du travail de Chakraborty et *al.*[5].

3.1 Modélisation de la puce sous forme de graphe

La puce DiffEq conçue par Intel a été initialement présentée dans le travail de Yun et *al.*[18] puis elle a été adaptée par Chakraborty et *al.*[5]. La Figure 19 présente l'architecture de la puce et le flot de données correspondant à une itération d'exécution des différentes opérations.

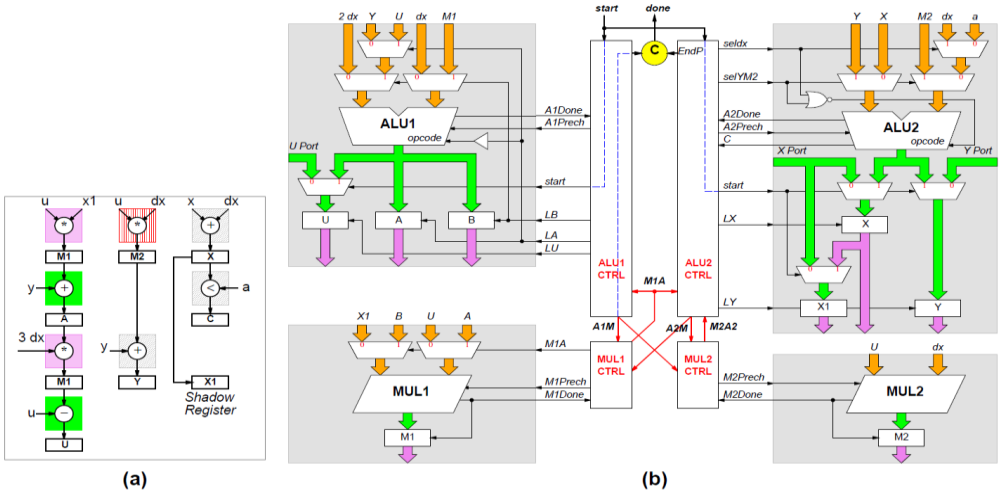


Figure 19(a) Graphe de flot de donnés de DiffEq (b) Architecture de la puce DiffEq[5]

Comme dans [5], lors de cette étude on s'intéressera au comportement itératif et opérations répétitives du flot de données ce qui correspond au gabarit du système stable de notre approche. Les opérations de fin de traitement lors de l'arrêt d'exécution doivent être étudiées séparément (en créant et étudiant le graphe acyclique correspondant).

Ce qui impliquera bien sûr que les résultats obtenus ne s'appliquent pas à la dernière itération qui représente la fin des calculs et l'arrêt d'exécution.

3.1.1 Description de la puce

Le contrôle de la puce est effectué à travers quatre contrôleurs XBM (Extended Burst Mode) : ALU1 CTRL, ALU2 CTRL, MUL1 CTRL et MUL2 CTRL. La communication entre ces contrôleurs est effectuée par le biais d'un protocole « handshake ». Les contrôleurs sont responsables de générer les différents signaux de : code d'opérations, sélection de multiplexeurs, changement des registres, chargement (load), évaluation (precharge) et activation (enable). Contrairement à certains systèmes asynchrones, les contrôleurs XBM se basent sur des hypothèses implicites concernant les temps de déclenchement des signaux qui doivent être validés (hypothèses) afin de garantir le bon fonctionnement de la puce. Par exemple, le contrôleur suppose que les données entrées aux circuits domino de la Figure 20sont stabilisées après un certain délai avant d'être évalué.

3.1.2 Modélisation

Afin de modéliser et représenter la puce sous forme de graphe d'événements et de contraintes temporelles, on utilise les diagrammes d'états/transitions de chaque composant et contrôleur. En effet, ces composants sont considérés comme des boîtes noires qui reçoivent des événements en entrée et produisent des résultats et des signaux en sortie. Pour chaque état, on associe deux ensembles de transitions : des transitions sortantes et des transitions entrantes. Chaque transition est représentée par différents événements. En règle générale, chaque relation entre les événements peut être représentée par les différents types de contraintes de la manière suivante :

- Si le signal de sortie ne se déclenche qu'après que toutes les transitions des signaux d'entrée ont eu lieu ; alors, un événement de type max est créé, puis lié d'une part, aux événements correspondant aux transitions de signaux entrantes avec des arcs de délais nulle ([0,0]); et lié d'autre part aux événements correspondants aux signaux de sorties avec des arcs de délais de traitement correspondants (des exemples seront donnés pour mieux expliquer ce procédé).
- Sinon si, le signal de sortie se déclenche dès qu'une transition d'un signal d'entrée a lieu alors un événement de type `min` est créé de la même manière décrite précédemment.

Ce procédé de modélisation est utilisé pour modéliser les contrôleurs et les différents composants de la puce comme les registres, les bascules et les bus. Après la modélisation de chaque composant, on regroupe les différents sous-graphes obtenus en utilisant le même procédé pour

modéliser les relations (les signaux de communications) entre les différents composants pour obtenir à la fin un graphe modélisant une itération complète représentant le comportement de la puce et les interactions entre les différents composants.

Dans ce qui suit, on donnera des exemples sur la modélisation de quelques-uns de ces composants qui serviront à mieux illustré le processus de modélisation décrit ci-haut.

3.1.3 Modélisation des contrôleurs

Comme indiqué précédemment, les contrôleurs sont considérés comme des boîtes noires qui ont des entrées et des sorties. Ce niveau d'abstraction permet de simplifier et faciliter le processus de modélisation tout en fournissant assez d'informations pour l'analyse de performances et la vérification temporelle de différents composants de la puce. Comme exemple, la Figure 20 illustre le diagramme de transition correspondant au contrôleur MUL2, ainsi que son modèle correspondant sous forme de graphe d'événements dans la Figure 20.b.

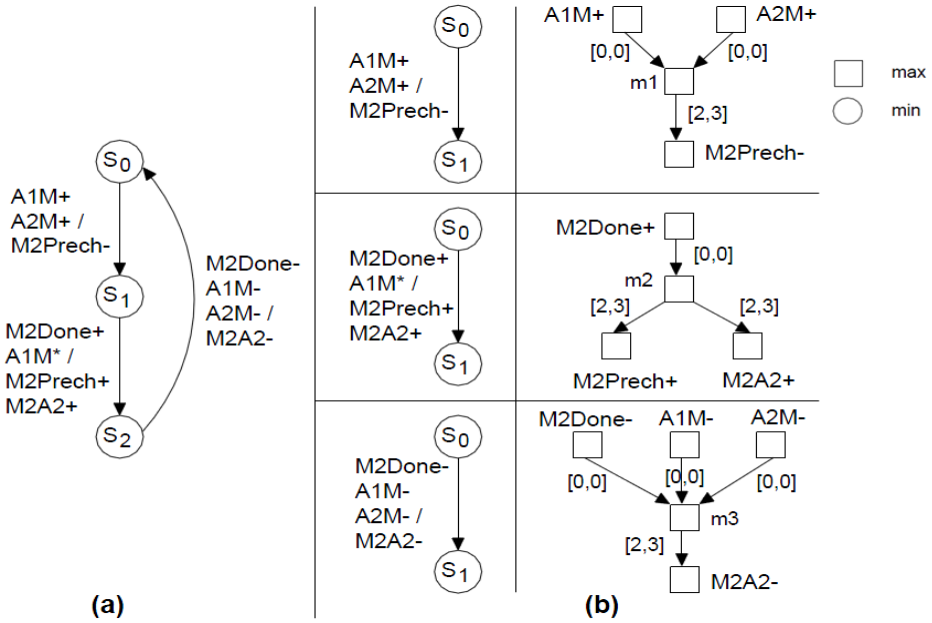


Figure 20 Modélisation du contrôleur Mul2 (a) Contrôleur Mul2 (b) Fragments du graphe de contraintes de Mul2 (avec délais traitement du contrôleur [2,3])[5]

Dans cette figure le signe '+' indique un front montant, le '–', front descendant et le '*' représente le « don't care ». Par exemple, de l'état S_2 à S_0, A1M a un front descendant, alors le

contrôleur doit attendre que le signal soit remis à zéro avant de passer de l'état S2 à S0, ceci est représenté (modélisé) par un arc reliant A1M à m3 dans la Figure 20.b.

Par contre, entre l'état S1 et S2 aucun arc ne relie A1M à m2 puisque A1M est un « don't care ».

Avec ce procédé, chaque contrôleur XBM est modélisé en formant un sous graphe d'événements. Puis, de la même manière, les relations entre les différents contrôleurs sont modélisées en reliant les entrées de l'un aux sorties des autres, comme c'est le cas de l'événement A2M entre l'état S2 et S0 qui est un événement de synchronisation émis par le contrôleur ALU2 à destination du contrôleur MUL2. De cette manière, en regroupant les différents sous graphes correspondants aux différents contrôleurs ensembles, on obtient un graphe modélisant les différentes interactions inter et intra contrôleurs représentant une itération complète de toutes les opérations la puce.

Un autre exemple est donné dans la Figure 21qui représente l'unité ALU2 (unité arithmétique logique) implémentée en logique domino. Le signal A2DONE indique que tous les nœuds internes sont pré-chargés s'il est mis à zéro (c.-à-d. A2DONE-) ou que l'évaluation est terminée s'il est mis à un (A2DONE+). La valeur du signal « opcode » détermine le type de l'opération de l'ALU (addition ou soustraction). Quand le signal A2PRECH+ a lieu, les sorties sont mises à zéro ainsi que A2DONE après un certain délai. Ceci est modélisé en liant A2PRECH+ directement à A2DONE- comme indiqué dans la Figure 21.b. Dans la Figure 21.a, on remarque qu'il y a une porte Non-Ou (NOR) entre les « in1 », « in2 » et « opcode », ce qui implique que dès que l'un des deux signaux entrants « in1 » et « in2 » est mis à un, le signal « opcode » est mis à zéro; d'où l'utilisation du nœud de type `min` dans la Figure 21.b pour refléter cette logique.

Le processus d'évaluation commence dès que A2PRECH est mis à zéro. Ici, le contrôleur suppose que les opérandes un et deux ainsi que le signal « opcode » sont stables avant de commencer l'évaluation. C'est-à-dire quel signal InputStable est supposé avoir lieu avant A2PRECH- (ceci doit être validé après lors de la vérification temporelle sinon il y aura une violation). Après l'évaluation, le signal A2DONE est mis à un indiquant ainsi la fin du calcul après un certain délai ceci est illustré dans la Figure 21.b.

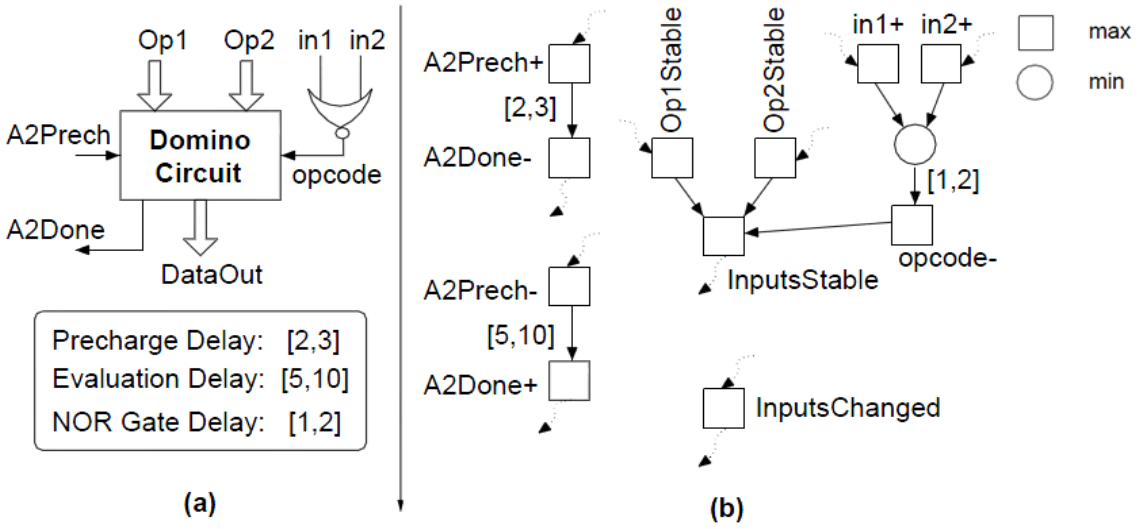


Figure 21 Modélisation d'un circuit domino (a) Graphe de contraintes et événements correspondant (b)[5]

3.1.4 Modélisation des registres

Après avoir étudié la modélisation de quelques contrôleurs, on donne maintenant un exemple sur la modélisation des registres. La Figure 22représente une bascule D actif sur front montant de l'horloge. Lors d'un front montant désigné par le signal LA dans les Figure 22 et Figure 19, les données en entrée sont reproduites en sortie après un certain délais. Les signaux DataInStable et DataIn change désignent respectivement l'étape hold et setup (la stabilisation des entrées). Le signal DataOutValid indique que la sortie est produite après un front montant désigné par le signal LA+, cette relation est représenté dans la Figure 22 par un arc reliant LA+ à DataOutValid avec le délai de reproduction des données de l'entrée vers la sortie.

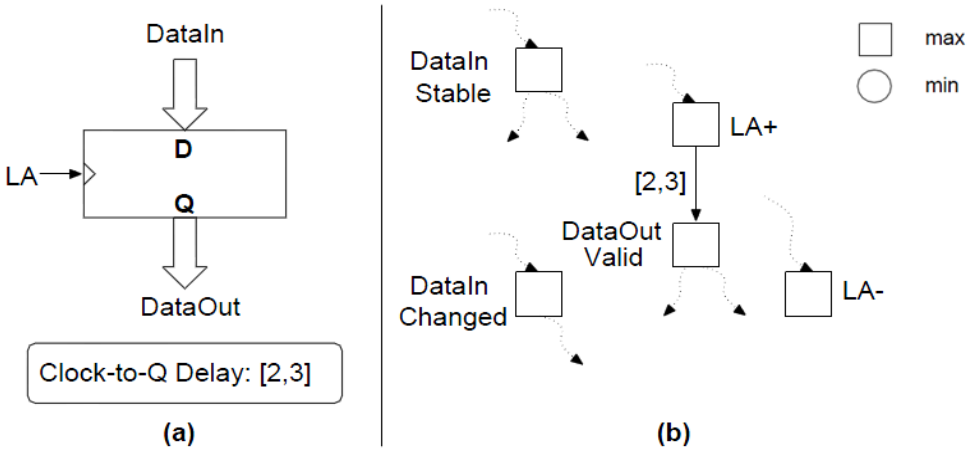


Figure 22 Modélisation d'une bascule D (a) Bascule D (b) Fragments de graphe de contraintes correspondants [5]

3.2 Vérification et validation des contraintes temporelles

Après avoir modélisé sous forme de graphe d'événements et contraintes les différents composants de la puce et regroupé tous ces sous-graphes en un seul représentant le déroulement des différentes opérations de la puce durant une itération, on passe à la vérification et la validation de contraintes afin de déterminer la consistance du système et s'il y a violation de ces contraintes au niveau des registres, contrôleurs, les signaux de chargement et stabilisation de données, etc. Le principe consiste à formuler et exprimer les exigences nécessaires pour le bon fonctionnement de la puce sous forme de temps de séparation. Par exemple, au niveau des registres, il faut s'assurer que les données sont stables avant qu'il y ait un front montant de l'horloge. Afin de satisfaire cette exigence, il faut vérifier si le minimum des temps de séparation entre la stabilisation des donnés et le front montant de l'horloge dépasse le temps que met le registre pour stabiliser les données (setup-time) c'est-à-dire la séparation entre les signaux LA+ et DataOutValid. De même, si on a besoin de déterminer la valeur minimale possible de la pulsation d'horloge, il suffit de calculer le minimum temps de séparation entre le front montant et descendant de l'horloge (c.à.d. entre LA+ et LA-). Un autre exemple est celui des circuits implémentés en « domino-logic » comme l'ALU dans la figure 11 responsable de l'addition ou la soustraction entre deux opérandes. Pour que cet ALU fonctionne correctement, il faut s'assurer que toutes les entrées (les opérandes et le code opération) soient stables avant que l'étape de l'évaluation commence. Afin de remplir cette exigence, il faut que le minimum des

temps de séparation calculés entre les signaux InputStable et A2PRECH soit positif, car une valeur négative signifierait que la valeur des données entrées peut être altérée ou changée pendant ou après la phase d'évaluation ce qui risque d'engendrer de faux résultats.

Comme dernier exemple, on va reprendre le contrôleur MUL2 de Figure 20. Dans la Figure 21.b, pour passer de S_0 à S_1 il faut que A1M et A2M soient mis à un, puis pour passer de S_1 à S_2 il faut que A2M reste à un jusqu'à ce que M2PRECH et M2A2 soient activés. Ce qui implique que le minimum temps de séparation de M2A2 et M2PRECH à A2M- doit être positif.

Après avoir modélisé la puce sous forme de graphe et formuler les exigences en fonction des temps de séparation, il ne reste plus qu'à les calculer et les comparer aux valeurs requises, de cette manière on vérifie si le système répond aux exigences, s'il est consistant et s'il y a des violations on peut localiser à quel niveau elles se sont produites.

3.3 Analyse et discussion des résultats

Nous avons repris le graphe décrivant le modèle de DiffEq du travail de Chakraborty et *al*. qu'ils ont construit à la main. Vu que la méthode de Chakraborty *et al.*[5] est basée sur le dépliage, on a d'abord commencé par le repliage du graphe pour passer de 688 (ça correspond à quatre dépliages) à 176 événements (nœuds) tous de types `min` ou `max` vu que leur méthode ne traite pas le cas linéaire. Puis, nous l'avons transformé sous forme de problème d'optimisation MILP, ensuite on l'a passé au solveur pour avoir les résultats. Les délais des composants ont été pris du simulateur SPICE sous des conditions typiques (22 C, 3.3V). Puis pour obtenir des intervalles avec des bornes supérieures et inférieures, un pourcentage de variation est introduit, dans notre cas c'était 10%. Pour les flots de données, le pire et le meilleur cas ont été pris en considération pour déterminer les délais correspondants [5].

On a obtenu en quelques secondes les mêmes temps de séparation que [5]. Nous avons réussi à détecter le bug que le simulateur SPICE a mis des heures à détecter lié aux délais restreints du buffer qui contrôle le signal A1M et qui peut causer le mal fonctionnement des circuits domino. Le tableau 1 donne quelques résultats des temps de séparation calculés et la ligne en gras représente une violation de contraintes.

TABLEAU 5 TEMPS DE SEPARATION REQUIS ET OBTENUS APRES CALCULS [5]

	Event1 (e_1)	Event2 (e_2)	Required min. ($t_2 - t_1$)	Computed $[d, D]$ $d \leq (t_2 - t_1) \leq D$	Timing Violations?
Register Timing	A1done+ LA+	LA+ A1done−	Setup-time (t_{su}) Hold-time (t_h)	[1.1250, 1.3750] [3.1970, 4.4630]	None if $t_{su} < 1.1250$ None if $t_h < 3.1970$
Domino Logic Timing	DataStable_M1	M1prech−	0	[0.1930, 3.2330]	None
	DataStable_A1	A1prech−	0	[0.6750, 1.7560]	None
	DataStable_A2	A2prech−	0	[1.8920, 4.0680]	None
	*DataStable_M1	M1prech−	0	[−0.7070, 3.2330]	**Potential violation**
Controller Protocol Timing	M2prech+	A2M−	0	[2.0430, 11.5430]	None
	M2A2+	A2M−	0	[2.3670, 11.7050]	None
	M2prech−	A1M−	0	[14.8870, 25.2550]	None

*violation au niveau de buffer contrôlant le signal A1M, les délais sont en ns)

4 Discussion et comparaison des résultats

On a réussi à avoir les mêmes temps de séparation que [5]en moins de quatre secondes, en utilisant un seul thread et moins de deux secondes avec deux threads (en divisant le nombre de séparations à calculer sur le nombre de threads désiré) avec une machine dotée d'un processeur i7, 2.8Ghz et 8 Go de RAM.

Comparé au travail de Chakraborty *et al* [5] et leur algorithme approximatif qui a mis 1.2 secondes et qui est jusqu'à maintenant considéré comme une référence pour l'approche analytique de systèmes cycliques et sur lequel d'autres travaux récents se sont basés, notre approche donne des temps de séparations exacts, (pourvu que Cplex donne le bon résultat) et sans faire de dépliage donc il est indépendant de la convergence du système. En plus, comme ils ont indiqué dans leur article, leur algorithme peut donner des résultats inexacts vu qu'il est approximatif, surtout s'il n'arrive pas à converger avant le nombre maximal de dépliages K_{max} (pour éviter de déplier à l'infini, ils ont fixé un nombre maximal de dépliages noté K_{max}, si le système ne converge pas avant K_{max}alors, les résultats vont être erronés). D'ailleurs, pour vérifier si leurs résultats sont corrects ou non, ils ont utilisé leur algorithme exact [6] pour le vérifier et qui a mis 11 heures sur leurs plateformes de test face à 1.37s pour l'algorithme approximatif sur la même plateforme [5]. En plus de ces améliorations que nous avons apportées à l'approche analytique, notre approche permet d'étudier des systèmes avec différentes hauteurs et elle traite aussi les contraintes linéaires et non seulement les `min` et `max` contrairement aux travaux précédents jusqu'à maintenant. LeTABLEAU 6donne une comparaison et les temps obtenus avec différentes approches.

TABLEAU 6RESULTATS OBTENUS AVEC DIFFERENTS GRAPHES ET COMPARAISON AVEC QUELQUES TRAVAUX PRECEDENTS

Nb d'événements	**Nb de contraintes Min-Max**	**Nb de séparations calculées**	**Temps Cplex**	**Temps avec d'autres approches**
120	17	120	1s	15 min[11]
152	37	152	3.8s	Plus de 24 heures*[11]
688 DiffEq (acyclicunfolded)	688	1376	2.4 min(1.4 avec 2 threads)	35 min[5]** 1.2s[5]***
176 DiffEq (foldedcyclic)	176	176	3.3 min (3.3 s avec période pré calculée)	Pas de travaux précédents qui traitent ce type de graphe

* On l'a laissé tournerpendant plus de 24 heures sur nos machines et toujours pas de résultats

**Algorithme CAS avec algorithme exacte Mc&Dill

*** Algorithme CAS avec ApproxAcylic[5]

Chapitre VI Conclusion

1 Conclusion

Au cours de ce mémoire, nous avons présenté une nouvelle approche pour le calcul des temps de séparations en utilisant une formulation MILP du problème permettant de modéliser et couvrir une plus grande variété de systèmes numériques que tous les travaux précédents effectués jusqu'à maintenant, puisqu'elle traite non seulement les systèmes acycliques mais les systèmes cycliques aussi avec tous les types de contraintes (min max linéaire) et ayant différentes hauteurs sans faire de dépliage. Les expériences effectuées sur des graphes aléatoires et sur des systèmes réels tels que la puce asynchrone de calcul d'équation différentielle ont permis de prouver et mettre en évidence l'efficacité et l'utilité de notre approche. Dans des travaux futurs, des améliorations peuvent être effectuées afin d'augmenter les performances du solveur Cplex et réduire les temps de calcul des temps de séparation en utilisant des fonctionnalités poussées du solveur pour ajuster et adapter la méthode de résolution générale qu'il utilise au problème d'analyse temporelle. La formulation qu'on a utilisée a permis aussi de résoudre d'autres problèmes connexes liés à l'analyse temporelle comme celui de l'ordonnancement des opérations des systèmes cycliques [6]. Notons aussi que puisque l'approche analytique des contraintes temporelles est indépendante des langages et/ou méthodes de conception et de niveau d'abstraction et puisque notre formulation couvre une grande variété de systèmes, elle peut être utilisée dans les différentes étapes et niveaux de conception de systèmes numériques. Par exemple, elle peut être utilisée à un niveau d'abstraction très bas, au niveau des registres et portes logiques pour étudier et vérifier le « setup and hold time » ou bien à un haut niveau d'abstraction pour vérifier les différentes interactions entre des blocs de propriété intellectuelle en les considérants comme des boîtes noires.

Notre travail met en évidence l'importance de l'approche analytique pour les problèmes de vérification temporelle comme alternative aux techniques de vérification coûteuses telles que la simulation, surtout pour les systèmes complexes de grande taille. Le domaine de la vérification

temporelle par l'approche analytique des contraintes temporelles reste encore un vaste domaine de recherche et sujet à de nombreuses améliorations surtout pour les systèmes cycliques.

2 Limite et travaux futurs

La tâche de construction de graphes à partir d'une description RTL (Registrer Transfer Layer) est reconnue comme étant une tâche fastidieuse et source d'erreur comme indiqué dans [5], étant donné que les graphes sont construits à la main. D'ailleurs, c'est l'une des raisons pour lesquels on a repris leur exemple de DiffEq vu qu'ils ont déjà généré le graphe correspondant, malgré qu'ils ne traitent pas le cas linéaire mais uniquement les contraintes de types `min` et `max` et la hauteur est limité à zéro ou un, mais ça nous a permis de comparer nos résultats avec d'autres approches sur un exemple réel et non trivial. En plus, pour le problème du calcul du maximum temps de séparation pour des systèmes cycliques, on n'a trouvé aucun travail antérieur avec des hauteurs supérieures à un, ce qui nous a obligés à utiliser des graphes triviaux ou générés de façon aléatoire pour faire les tests et expérimentations.

Comme travaux futurs, on pense qu'une étude plus approfondie du problème de temps de séparation afin de générer des plans de coupes et des colonnes de manière plus efficace et spécifique au problème étudié, permettra d'ajuster et d'adapter l'algorithme utilisé par le solveur Cplex(*branch & cut*) afin d'augmenter ses performances et réduire considérablement le temps de calcul des temps de séparation.

Aussi, on peut encore chercher à étendre notre formulation afin de couvrir d'autres types de contraintes comme les contraintes « Assume » et « Commit » expliquées dans [11].

Bibliographie

[1] Nascentric. (2010, Avril) HPC Wire. [Online]. http://archive.hpcwire.com/hpc/2280791.html

[2] T. M. Burks, "Timing verification and optimization of circuits with level sensitive latches", University of Michigan, Thése de doctorat 1994.

[3] A. J. Gahlinger, "Coherence and Satisfiability of Waveform Timing Specifications", Université de Waterloo, Waterloo, Ontario, Canada, Thése de doctorat 1990.

[4] K. L. McMillan et D. L. Dill, "Algorithms for interface timing verification", *Computer Design: VLSI in Computers and Processors, 1992. ICCD '92. Proceedings., IEEE 1992 International Conference*, pp. 48-51, Octobre 1992.

[5] S. Chakraborty, K. Y. Yun, et D.L. Dill, "Timing analysis of asynchronous systems using time separation of events", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions*, pp. 1061 - 1076, Août 1999.

[6] I. E. Bennour et E. M. Aboulhamid, "Les problèmes d'ordonnancement cycliques dans la synthèse de systèmes numériques", DIRO, Université de Montréal, Rapport technique 996, Oct. 1995, http://www.iro.umontreal.ca/~aboulham/pipeline.pdf.

[7] A. Azzabi, E. M. Aboulhamid, et G. Nicolescu, "Timing verification of cyclic systems based on temporal constraint analysis", *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference*, pp. 659 - 662, Décembre 2010.

[8] P. Vanbekbergen, G. Goossens, et H. De Man, "Specification and analysis of timing constraints in signal transition graphs", *Design Automation, 1992. Proceedings, European Conference*, pp. 302 - 306, Mars 1992.

[9] T. Y. Yen, A. Ishii, A. Casavant, et W. Wolf, "Efficient Algorithms for Interface Timing Verification", *Formal Methods In System Design*, vol. 12, no. 3, pp. 241-265, Avril 1998.

[10] F. Brglez, D. Bruan, et K. Kozminski, "Accelerated ATPG and Fault Grading via Testability Analysis," *IEEE International Symposium on Systems and Circuits*, pp. 695-698, Juin 1985.

[11] A. Tsikhanovich, E. M. Aboulhamid et G. Bois, "Timing specification in transaction level models", *System Level Design with.Net Technology*, pp. 209-244, Séptembre 2009.

[12] K. K. Parhi et D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding", *Computers, IEEE Transactions*, vol. 40, no. 2, pp. 178-195, Février 1991.

[13] T. Amon, H. Hulgaard, G. Borriello, et S. Burns, "Timing analysis of concurrent systems", Univ. Washington, Seattle, Rapport Téchnique UW-CS-TR-92-11-01, 1992.

[14] Q. Zha, J. Mao , et T. Ye, "Time separations of cyclic event rule systems with min-max timing constraints", *presented at Theor. Comput. Sci.*, pp. 496-510, 2008.

[15] Wikipédia. [Online]. http://fr.wikipedia.org/wiki/D%C3%A9composition_LU

[16] Support IBM ILOG. IBM ILOG CPLEX Optimization Studio, Information Center. [Online]. http://publib.boulder.ibm.com/infocenter/cosinfoc/v12r2/index.jsp

[17] W3C. Extensible Markup Language (XML). [Online]. http://www.w3.org/XML/

[18] K.Y. Yun, P.A. Beerel, V. Vakilotojar, A.E. Dooply, et J. Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver", *Very Large Scale Integration (VLSI) Systems, IEEE Transactions*, vol. 6, pp. 643-655, Décembre 1998.

[19] T.M. Burks et K. Sakallah, "Min-max linear programming and the timing analysis of digital circuits", *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference*, pp. 152-155, Novembre 1993.